Dark BASIC Professional
PROGRAMMING MANUAL

CONTENTS

DARK BASIC PROFESSIONAL TEAM CREDITS

DarkBASIC Professional Team
Lee Bamber...Compiler Programmer
Mike Johnson...3D Technology Programmer
Guy Savoie...Editor Programmer
Rick Vanner...Project Manager
Richard Davey...Website
Simon Benge...Lead Artist

Support Team
Malcolm Bamber...Programming and Ideas
Christopher Bamber...Artist
Darren Ithell...Music and sounds effects
MONOS...Gwain

Special Thanks
The DBDN Members...Testing and Ideas
The DarkBASIC Community...Ideas

INTRODUCTION

Welcome to DarkBASIC Professional
The Ultimate Programming Language for your PC!

No matter what type of game you plan to make, by providing rapid development solutions, Dark Basic Professional has the power to handle them all. It does all this and remains the easiest programming language available.

The original Dark Basic was designed in 1999 to facilitate the easy programming of 3D computer games on the PC. The combination of ease and power was an instant hit, and a vibrant and creative community grew up around the language. With the emergence of new technology and techniques, it was clear the evolution of the language should race to meet these new advances. After two years of development, Dark Basic Professional is born.

For those of you who have jumped right in, you will want to know where to go first. From the introduction help page, we suggest you go to the main menu and select GETTING STARTED. From there you will be able to gain a insight into the various parts of the software. It should only take around fifteen minutes to familiarise yourself with the editor. After that, we suggest you check out some of the showcase applications in the EXAMPLES section.

GETTING STARTED

These lessons will guide you through the various tools you will need to make the most of the language. They have been designed to give you the most information in the shortest time. They will only take a few minutes to complete, and will prepare you for your continued use of the language.

Installing The Software

1. Insert the CD into the CD-Drive. If your computer has autoboot, wait for the splash screen to appear. If your computer does not have autoboot, go to My Computer, select the CD Device containing the Dark Basic Professional CD, right click the item and select Autoplay.

2. From the Boot Menu Selection Screen, click Install Dark Basic Professional to install the software onto your computer.

3. When the installation is complete, you will find a Dark Basic Professional icon has been created on your desktop. To run the software, double click this icon.

Note: Direct X8.1 MUST be installed on your system for DarkBASIC Professional to work properly (no lower version of Direct X will do).

Understanding The User Interface

The user interface is shown below and divides into the main editing area, the menu system, the status bar and an optional side panel.

The menu bar divides into the following options:

FILE Menu

NEW PROJECT : Starts a new project
OPEN PROJECT : Opens an existing project
OPEN SOURCE : Opens an existing DBA file for current project
REOPEN : Opens a project previously worked on
SAVE PROJECT : Save the current project
SAVE PROJECT AS : Save project under a new name
SAVE SOURCE : Save currently edited source code
SAVE SOURCE AS : Save current source code under a new name
SAVE ALL : Save all source code of current project
CLOSE : Close window holding current source code
CLOSE ALL : Close all windows containing source code
PRINT : Print current source code to printer
EXIT : Exit the editor

EDIT Menu

UNDO : Restore source code to previous state
REDO : Reapply state prior to UNDO being used
CUT : Copy and cut currently selected text to clipboard
COPY : Copy currently selected text to clipboard
PASTE : Paste contents of clipboard to source code cursor
DELETE BLOCK : Deletes all text currently highlighted
SELECT ALL : Selects all text in current source code window

TOGGLE BREAKPOINT : Toggles breakpoint within source code
SET BOOKMARK : Sets a bookmark at the current cursor position

SEARCH Menu

FIND : Finds a text match within the current source code
FIND AGAIN : Uses text from previous search to find again
SEARCH/REPLACE : Finds text and replaces it with alternative
GO TO LINE : Jumps to a specified line number
GO TO BOOKMARK : Jumps to a specified bookmark position

VIEW Menu

SHOW TOOL BAR : Shows/Hides the iconic shortcut toolbar
SHOW PROJECT MANAGER : Shows/Hides the project settings
TOOLBARS : Control toolbar visibility
DISPLAY MODE : Controls editor mode
FOLD ALL FUNCTIONS : Folds each function to a single line
UNFOLD ALL FUNCTIONS : Unfolds all functions to normal

COMPILE Menu

CHECK SYNTAX/MAKE EXE : Compiles current project
MAKE EXE/RUN : Compiles and runs current project
RUN IN DEBUG MODE : Compiles and runs in debug mode
RUN IN STEP-THROUGH MODE : Compiles and runs in step-through mode

TOOLS Menu

SYSTEM OPTIONS : Shows system information

WINDOW Menu

CASCADE : Arranges windows in cascade order
TILE HORIZONTALLY : Arranges windows in horizontal order
TILE VERTICALLY : Arranges windows in vertical order

HELP Menu

CONTENTS : Shows help at contents page
LAST HELP FILE : Shows last help page viewed
INDEX : Shows command index page
CHECK FOR UPGRADES : Checks DarkBasic Website for Upgrades
DARK BASIC PROFESSIONAL HOME : Go to Dark Basic Professional Website
DARK BASIC DEVELOPER NETWORK : Go to DarkBASIC Developer Network Website
DARK BASIC SOFTWARE HOME : Go to Dark Basic Software homepage
ABOUT : Shows software about dialogue

LESSON 1
Using The Language

1. Familiarise yourself with the large blank area known as the editing area.

2. Click into this area, then type the following:

```
PRINT "HELLO WORLD"
WAIT KEY
```

3. This is a simple program to display "HELLO WORLD" to the screen, then wait for the user to press a key before exiting. To run the program, you can either press F5, use the COMPILE Menu or click the iconic shortcut that looks like a green play button:

LESSON 2
Using The Debugger

1. Type into the editing area:

```
DO
T=T+1
PRINT T
LOOP
```

2. The debugger allows you to watch your program while it runs. This can be useful to track down why your program does not behave as you expected. To active the debugger, press ESCAPE when you run the program. To run the program in debug mode, click the Debug Button from the Menu Area:

3. Press the ESCAPE Key to make the Debugger Window appear, then select "Step Through Mode" from the menu bar and then "Slow Program" from the available buttons. This slows the program down and allows you to watch your program instructions being executed one at a time:

4. To watch the state of your programs variables, select "Variable Watch" and observe the variable T slowly increment. You can observe any global variable being used by your program. By selecting the variable from the Debugger Window, you can store it in a watcher window. This is useful when the number of variables in your program exceeds the size of the watcher view.

5. To close the debugger, click the Close Button located in the top right corner of the Debugger Window:

LESSON 3
Using The Help System

1. Select the Help menu item from the menu bar, and click Contents. The editor allows multiple help windows open at the same time. This allows you to reference both the help guides and reference pages simultaneously.

2. Select Examples from the Main Menu in the newly opened help window.

3. Select the Example; Animation Showcase.

4. An icon in the help browser menu bar will be flashing, which will load the example into the editor when pressed. Press this icon:

5. Press F5 to run the example program, and then ESCAPE to quit.

6. Select the 'GO BACK' button twice to return to the Main Menu:

7. Click the close button on the help window to close the help.

LESSON 4
Using The Project Panel

1. The project panel is used to control the finer aspects of the executable you will ultimately produce to distribute your finished program. Select the view menu from the menu bar, then click Project View from Display Mode item.

2. After confirming the change in editor mode, select File from the menu bar, and click New Project.

3. The Project panel is divided into sections as detailed below.

4. The project summary is the header for your project and details the general information about your project and what you plan to do. To give your project a name, click the Project Name box and type in a name.

5. From the base of the project panel, select Settings and choose to Compress the final executable.

6. Then switch to the Media section, and click Add to include a picture file into your project.

7. Switch to the Cursors section, and click Add to include an image for your application cursor.

8. Switch to the Icons section, and click Add to include an image for your application icon.

9. To build your final executable as described in the EXE Filename box of the Project Summary section, click the Build EXE icon at the top of the screen:

LESSON 5
My First 3D Program

1. Creating a 3D game or application is a simple matter of adding levels of detail to a program. Take a simple cube spinning program. The steps are to first create the cube, then to turn it slowly over time.

2. Type out the following small program into the editing area:

```
MAKE OBJECT CUBE 1,100
DO
 YROTATE OBJECT 1,OBJECT ANGLE Y(1)+0.1
LOOP
```

3. This program will loop forever spinning the cube very slowly. The MAKE OBJECT CUBE command first creates the cube, the DO and LOOP commands create the infinite loop and the YROTATE OBJECT command applies a new rotation angle to the cube. The new rotation angle is calculated from the current angle of the object plus a small incremental value. As the loop goes round and round, the angle is slowly incrementing causing the cube to spin.

Migrating From Dark Basic V1

For those users who are moving their project over to Dark Basic Professional, or have upgraded and want to get up and running as quickly as possible, the following tips may help your transition.

Command Line Interface

The CLI is now part of the Debugger and can be located by clicking the Debug Mode Button in the main editor. You will find many of the reasons for having a CLI have been replaced with more powerful Variable Watcher and Program Watcher tools.

Desktop Full screen Standard

The default running mode for all Dark Basic Professional programs is FULLSCREEN DESKTOP. This means the application runs in windows mode stretched to the size of the desktop. This is a preferred method of rendering with modern hardware and WindowsXP. Your framerates are faster and you have full cross-application ability throughout the execution of your program. You still have the option for full screen exclusive mode, as well as hidden mode, standard window mode and a special fullscreen desktop mode which retains the taskbar in case you are writing a tool rather than an a game which ideally plays into the entire screen. Be aware some older cards incur a performance penalty when running in full screen desktop mode.

Obsolete Commands

Approximately thirty commands have been discontinued from the language. There are various reasons for this, mainly to ensure that the language takes advantage of the best methods of achieving the results you desire. A detailed list follows.

Four BASIC3D commands have been removed, found to be redundant in the latest implementation of the language:

3DS2X

ENABLE TNL

DISABLE TNL

SET MIPMAP MODE

Five BASIC3D animation commands have been removed, to make way for future enhancements providing much better support for all types of 3D object animation:

APPEND OBJECT ANIMATION

CLEAR ALL OBJECT KEYFRAMES

CLEAR OBJECT KEYFRAME

SAVE OBJECT ANIMATION

SET OBJECT KEYFRAME

Eleven BASIC3D static object commands have been removed, to shift support to the new built-in BSP and Nodetree engines for world construction:

LOAD STATIC OBJECTS

MAKE STATIC OBJECT

DELETE STATIC OBJECTS

SAVE STATIC OBJECTS

ATTACH OBJECT TO STATIC

DETACH OBJECT FROM STATIC

DISABLE STATIC OCCLUSION

ENABLE STATIC OCCLUSION

SET STATIC OBJECTS TEXTURE

SET STATIC OBJECTS WIREFRAME OFF

SET STATIC OBJECTS WIREFRAME ON

One BITMAP command has been removed, to make way for future enhancements providing much better support for all types of media exporting:

SAVE BITMAP

Three DISPLAY commands have been removed, due to redundancy:

EMULATION MODE
SET EMULATION OFF
SET EMULATION ON

One IMAGE command has been removed, due to redundancy:

ROTATE IMAGE

One SOUND command has been removed, to make way for future enhancements providing much more control over Environmental Audio Effects using the very latest technology:

SET EAX

PRINCIPLES

The purpose of this section of the help is to teach you how to write your own programs. BASIC is an acronym for Beginners All Purpose Symbolic Instruction Code. DARK derives from the term 'Dark Horse'. The traditional description of a program is a task that you want your computer to perform. The task is described to the computer using statements the language can understand.

Statements in your program must be written using a set of rules known as 'Syntax'. You must follow these rules if you are to write programs. By proceeding through these sections in sequence, you will gain a firm understanding about the general rules of BASIC and how to apply them as a programmer.

Data Types, Variables and Arrays

DATA TYPES

We have established that statements are used to write a program. A statement can be broken up into a command and its data. The command is the operation, or task you wish to perform. The data is that which must be used by the command to complete the operation. The data is also referred to as the parameter(s).

There are many types of data you can use, including integer numbers, real numbers and string. Each type of data holds a slightly different type of value.

Integer Numbers

An integer number can hold a whole number, but no fraction. For the value to be negative, you must place a hyphen symbol (-) before the value. You must not use commas as part of the number as this would generate a Syntax Error. Examples of integer numbers:

```
42
10000
-233000
-100
```

Real Numbers

A real number can hold a whole number or a fractional number that uses a decimal point. For the value to be negative, you must place a hyphen symbol (-) before the value. Examples of real numbers:

```
20.0005
99.9
-5000.12
-9999.9991
```

Strings

String data is non-numerical, and is used to store characters and words. All strings consist of characters enclosed within double quotation marks. The string data can include numbers and other numerical symbols but will be treated as text. Examples of strings are:

```
"A"
"Hello World"
"Telephone"
"I am 99 years old"
"1.2.3.4.5.6.7.8.9"
```

Each string can consist of as many characters as the memory allows. You can also have a string with no data whatsoever, represented by an empty pair of double quotation marks.

DATA TYPE RANGES

Each type of data has a maximum and minimum value known as the range. It is important to know these ranges, especially when dealing with smaller datatypes. Below is a list of datatypes and their ranges:

```
INTEGER Range : −2,147,483,648 to 2,147,483,647
REAL Range : 3.4E +/- 38 (7 digits)
BOOLEAN Range : 0 to 1
BYTE Range : 0 to 255
WORD Range : 0 to 65535
DWORD Range : 0 to 4,294,967,295
DOUBLE INTEGER Range : −9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
DOUBLE FLOAT Range : 1.7E +/- 308 (15 digits)
```

VARIABLES

The best way to demonstrate what a variable does is by way of an example. Take the calculation:

```
A = 3 + 4
```

A variable is used to store a value. It's that simple. You can have a variable that stores any type of data, and you can have as many as you want. The following program shows you how the contents of the variable can be output to the screen:

```
A = 3 + 4
PRINT A
```

Now take the next example to show you how variables can be used as freely as standard number types:

```
A = 2
B = 8
C = A + B
PRINT C
```

In the preceding example, 3 is stored in the A variable, 4 is stored in the B variable and C is given the result of the calculation between A and B. The calculation is based on the values stored within the variables and so the calculation is actually C = 2 + 8. The result, which in this case is 10, is stored as the new value of C and this is the value that eventually gets printed to the screen.

So far, we have seen variables used to store and recall integer values. Variables can also store real numbers and strings. In order to allow a variable to store these other types of data, you must make sure the variable is recognized as an integer, real or string variable. To name a real number variable, you must add a hash character (#) as the last character of the variable name. If you want your variable to store a string, you must add a dollar character ($) as the last character of the variable name. Let's see these new variables used to store and recall real values:

```
mydata#=42.5
PRINT mydata#
```

By adding the (#) symbol, we are instructing the program to treat the variable as a real number variable. Exactly the same rule applies to a string variable:

```
myname$="Lee"
PRINT myname$
```

All variable names can use either upper or lower case characters, which means a variable called NAME$ is the same variable as name$ or Name$. String variables even support the use of limited maths. The following example adds two strings together and the result is a concatenation of the two strings:

```
a$="Hello"
b$="World"
c$=a$+b$
print c$
```

To run this example, the text "HelloWorld" will be printed to the screen. Would you be able to alter this example to place a space between "Hello" and "World"?

ARRAYS

Arrays are going to be a very important part of your future programs. They allow you to store large amounts of data under a single name. You can then access the data by index rather than by name alone.

If you had to write a program that stored each weeks lottery numbers, typing out 52 unique variable names is a lot of work, hard to maintain and quite unnecessary. Arrays allow you to create a special kind of variable that can store more than one item of data. You might start your program like this:

```
lottery1$="43,76,12,34,12,11"
lottery2$="76,12,34,12,11,44"
lottery3$="12,34,12,02,05,07"
etc..
```

Two hours later, you realize you could have written it like this:

```
DIM lottery$(52)
lottery$(1)="43,76,12,34,12,11"
lottery$(2)="76,12,34,12,11,44"
lottery$(3)="12,34,12,02,05,07"
etc..
```

We declare a string array using the DIM command followed by a name for our array. Like variables, when we use a dollar symbol after the name we instruct the program to use the array to store only strings. We then enclose in brackets how many items of data we wish the array to store. The array can be filled almost like a variable, but you must also provide the position within the array you wish to store your data.

But you then ask yourself what benefits I would have gained using the second approach. If you where also required to print out all 52 lottery numbers to the screen with your first approach you would have to add another 52 statements that printed each variable:

```
PRINT lottery1$
PRINT lottery2$
PRINT lottery3$
etc..
```

But if you had used an array, the same example would look like this:

```
PRINT lottery$(1)
PRINT lottery$(2)
PRINT lottery$(3)
etc..
```

You will have noticed that by using an array, you no longer have to refer to your data using a unique variable name. You can now point to the data you want using a position number. Accessing data this way has a thousand advantages over trying to access data by variable name alone, as you will discover. One example would be to improve the above like this:

```
FOR T=1 TO 52
PRINT lottery$(T)
```

```
NEXT T
```

Incredibly the above code replaced 52 PRINT statements with just 3 statements. With the above example, T is incremented from 1 to 52 within a loop that prints out the contents of the array at that position.

Arrays can also store multiple levels of data. At the moment our lottery entries are stored as strings and the numbers are hard to get at. Let's say we wanted to store all six numbers for every lottery week, we would create an array like this:

```
DIM lottery(52,6)
```

Without the dollar symbol($), we are declaring the array to store integer numbers instead of strings. You will also notice we have a second number separated by a comma. This means for every array position from 1 to 52, there is a sub-set numbered 1 to 6 in which multiple data can be stored. You can visualize an array as a filing cabinet with large draws numbered 1 to 52. Within each of the 52 draws is a tray with 6 boxes inside. You can store a value in each box. In all you can store 312 (52 x 6) values in this array. You can have up to five dimensions in your array, which means you can create an array as big as (1,2,3,4,5). Be careful when declaring dimensions, as large arrays consume large amounts of memory and may reduce overall performance of your program.

Entering data into our new array is elementary:

```
lottery(1,1)=43
lottery(1,2)=76
lottery(1,3)=12
lottery(1,4)=34
lottery(1,5)=12
lottery(1,6)=11
lottery(2,1)=43
lottery(2,2)=76
lottery(2,3)=12
lottery(2,4)=34
lottery(2,5)=12
lottery(2,6)=11
```

You are now able to give your program access to much more useful data. Unlike the string approach, you could make your program count how many times a certain number has appeared.

As you have determined, arrays need to be declared as a particular type. You can have an array of integer numbers, real numbers or strings. You cannot have multiple types in the same array, but you can declare new arrays dedicated to holding such data.

You can also declare arrays as global or local. Global arrays are the ones you are familiar with, and can be accessed by any part of the program. Local arrays can only be accessed by the function in which it was created. It is important to note that the global array must be delcared at the top of the main source code of the program as arrays are dynamically created only when the DIM command is executed. Placing DIM commands at the top of included source code will not dynamically create the array unless it lies within a subroutine called from the main program.

USER DEFINED TYPES

If the current set of datatypes is inadequate for your needs, you can can create your own data types using user-defined-type. User defined types are useful for storing data using logical fields rather than the unfriendly list of subscripts used by arrays.

To create a user defined type, you must first declare it at the top of your program. To do so, you would give your type a name and a list of fields it contains:

```
TYPE MyType
 Fieldname1
 Fieldname2
 Fieldname3
ENDTYPE
```

The above code creates a type called MyType with three fields contained within it. As the fields have no declaration, they are assumed to be integers. The same code could also be truncated to a single line like so:

```
TYPE MyType Fieldname1 Fieldname2 Fieldname3 ENDTYPE
```

To use your type, you simply create a variable and declare it with your new type. To declare a variable as a specific type, you would use the AS statement:

```
MyVariable AS MyType
```

You can then assign data to your variable as normal, with the added bonus of the fields you have given your variable like so:

```
MyVariable.Fieldname1 = 41
MyVariable.Fieldname2 = 42
MyVariable.Fieldname3 = 43
```

At the moment, the type is assuming our fields are integers. We may wish to declare our fields as a real number, string or other datatype. We can do so using the same AS statement within the type definition. So the following code makes more sense we shall give our type and fields sensible names:

```
TYPE AccountEntryType
 Number AS INTEGER
 Name AS STRING
 Amount AS FLOAT
ENDTYPE
```

You can use a type like any other, so creating and using an array of the above is simply a case of declaring the array with your new type:

```
DIM Accounts(100) AS AccountEntryType
Accounts(1).Number=12345
Accounts(1).Name="Lee"
Accounts(1).Amount=0.42
```

As you will eventually discover you can have types within types for more complex data structures so we can imagine one of the fields contains more than one value. We would define two user defined types, and then use one of them in the declaration of one of the fields of the section user defined type, as follows:

```
TYPE AmountsType
 CurrentBalance AS FLOAT
 SavingsBalance AS FLOAT
 CreditCardBalance AS FLOAT
ENDTYPE

TYPE AccountEntryType
 Number AS INTEGER
 Name AS STRING
 Amount AS AmountsType
ENDTYPE

DIM Accounts(100) AS AccountEntryType
Accounts(1).Number=12345
Accounts(1).Name="Lee"
Accounts(1).Amount.CurrentBalance=0.42
Accounts(1).Amount.SavingsBalance=100.0
Accounts(1).Amount.CreditCardBalance=-5000.0
```

As you can see, user defined types are not only powerful, they make the readability of your programs far easier. Using named fields instead of a subscript value within an array, you can save yourself many hours all for the sake of an incorrect subscript value throwing out your program results.

POINTERS

You can use the value of a variable to specify, read and write the contents of an address by using the '*' indirect symbol. This is useful for obtaining areas of memory you wish to read from or write to and control the pointer into this memory using a standard variable.

```
Ptr as DWORD
Ptr=0x00FF8820
*Ptr=42
PRINT *Ptr
```

The above code simply declares Ptr as a DWORD variable, assigns the variable a random area of memory, writes the value 42 into that location of memory and then reads the same address back, the contents of which is to be printed to the screen. You would never use an absolute address to assign a pointer. There are commands such as GET BACKBUFFER PTR and MAKE MEMORY which return such address values. You can only use a standard variable with indirection, and array and type variables are not valid. This is an advanced feature and as such potentially dangerous. Writing into invalid memory can cause unexpected program behaviour and even crashing. Use caution when employing the indirect symbol.

Arithmetic, Relational and Boolean Operators

We have already used one type of well-known operator in the preceding examples. Operators are the term given to a mathematical symbol used in all calculations. The most common operators are arithmetic operators and are quickly identified. All operators require two operands of data that are placed either side of the operator.

ARITHMETIC OPERATORS

An arithmetic operator can represent an Addition, Subtraction, Multiplication or Division. These operators are represented symbolically as (+) (-) (*) (/) respectively.

The Plus(+) sign specifies that the data on the right of the plus sign must be added to the data on the left. Examples of which you have already seen are:

```
3 + 4 equals 7
A + B equals the value of B added to the value of A
```

The minus(-) sign specifies that the data to the right of the minus sign must be subtracted from the data to the left of the minus sign:

```
3 - 4 equals -1
A - B equals the value of B subtracted from the value of A
```

An asterix(*) specifies that the data on the right side of the asterix is multiplied by the data on the left side if the asterix:

```
3 * 4 equals 12
A * B equals the value of B multiplied by the value of A
```

The slash(/) specifies that the data on the left side of the slash is to be divided by the data on the right side of the slash:

```
10 / 2 equals 5
A / B equals the value of A divided by the value of B
```

The MOD specifies that the data on the left side of the MOD is to be divided by the data on the right side of the MOD, and the remainder of the division is the result:

```
11 MOD 2 equals 1
A MOD B equals the remainder of the division between A and B
```

RELATIONAL OPERATORS

These operators are less common, unless you have programming experience. These operators represent conditions that are applied to data. The conditions handled are Equal To, Greater Than, Less Than, Greater or Equal To, Less or Equal To and Not Equal To. The purposes of these conditions are to determine the result of a comparison between two data values. A condition result can only be of two possible values. If the condition is false, the resulting value is zero. If the condition is true, the resulting value is one. Take the following examples:

```
10 = 9 results in 0 because 10 is not the same as 9
10 = 10 results in 1 because 10 is the same as 10
10 > 9 results in 1 because 10 is greater than 9
100 >= 100 results in 1 because 100 is greater or equal to 100
```

The same relational operators can be applied to real numbers, integer and real variables and in some case strings and string variables. You can compare whether two strings are the same or not the same, and even test whether one string is greater or less than another.

BOOLEAN OPERATORS

Simple Boolean operators provide the last type of operator. Dark Basic Professional allows you to use AND, OR, XOR and NOT operators on your data.

The AND operator works with any integer value, but for demonstration purposes the general rule applies when using this operator:

```
0 AND 0 = 0
0 AND 1 = 0
1 AND 0 = 0
1 AND 1 = 1
```

What you see is the decision tree of the AND operator. It shows that only if both data operands of the AND operator are 1 will the result be a 1 also. All other cases a zero is returned. To see how this logic works in reality, take the following example:

```
A=5
B=25
(A > 10) AND (B > 20) so what is the resulting value?
```

We can determine the result of the parts enclosed in brackets first. We can see the relational operators provide us with the following results:

```
(A > 10) results in 0 because 5 is not greater than 10
(B > 20) results in 1 because 25 is greater than 20
```

Our updated calculation looks something like this:

```
(0) AND (1) results in 0 as our table shows 0 AND 1 = 0
```

The logic of the table is that only when both sides of the AND operand are 1 will the result of the calculation be 1 also. What would happen if you change the value of A to 15?

The OR operator works in a similar fashion, but using the following table. If either the left side or right has a value of 1, the result will be 1:

```
0 OR 0 = 0
```

```
0 OR 1 = 1
1 OR 0 = 1
1 OR 1 = 1
```

The NOT operator works using the following table. This operator is a unary operator and only requires a single right-side value:

```
NOT 0 = 1
NOT 1 = 0
```

BITWISE OPERATORS

Bitwise operators, unlike boolean operators work on all the bits of the specified variable or value. There are six bitwise operators as follows:

BITWISE LEFT SHIFT signified by the symbol << will shift all bits one space to the left. %0111 << 1 becomes %1110.
BITWISE RIGHT SHIFT signified by the symbol >> will shift all bits one space to the right. %0111 >> 1 becomes %0011.
BITWISE AND signified by the symbol && will AND all bits of one value with another. %1111 && %0011 becomes %0011.
BITWISE OR signified by the symbol || will OR all bits of one value with another. %1111 || %0011 becomes %1111.
BITWISE XOR signified by the symbol ~~ will XOR all bits of one value with another. %1111 ~~ %0011 becomes %1100.
BITWISE NOT signified by the symbol .. will NOT all bits of the right value. %1111 .. %1010 becomes %0101.

You will discover how useful these operators become when writing conditions for your programs. Being able to write conditions with multiple parts will become increasingly important as you begin to write more complex programs.

Common Statements

ASSIGNMENT STATEMENTS

You have already used an assignment statement, and is probably the most commonly used part of any programming language. The Equal Symbol (=) is used to assign a value to a variable or array. Take the following examples:

```
a=42
a#=99.9
a$="HELLO"
lottery(1,1)=49
```

DATA AND READ STATEMENTS

There are many occasions where you will be required to store data inside your program. Take our earlier lottery example. It would be much better to make a list of numbers at the end of the program and simply add to the list as you get new lottery results. Using the DATA and READ commands you can do exactly that. Look at the following example:

```
DATA 9,"NINE",9.9
READ a,a$,a#
```

The DATA command accepts a list of data items separated by a comma. The data items do not have to be of the same type, but they do need to be read in the right type order. As you can see our first item of data is an integer number of 9, then a string with the text "NINE" and a real number with a value of 9.9.

The READ command also accepts a list, but the list must contain variables that are of the correct type to read the data. When an item of data is read, a pointer moves to the next item of data stored in the list. The first item of data is an integer, which means the value of 9 is stored in the integer variable of A successfully. The data pointer then moves to the next item that is the string that stored "NINE" text. This is read and stored in the string variable A$. The same applies to the real number data of 9.9.

If you were to visualize the data list in memory it would look like this:

```
9
"NINE"
9.9
```

If you tried to read the integer value of 9 into a string variable, an empty string will be stored as the types are incompatible. If you tried to read a string into an integer or real variable, a zero will be stored. It is your responsibility to make sure the type order used to store you data is the same when you come to read it back.

RESTORE STATEMENTS

You are able to reset the data pointer at any time using the RESTORE command. If for example you have read the data and printed it to the screen, you will need to read the same data again if the user wants to clear the screen and redraw the data. By resetting the data pointer the READ command will start at the top of the data list and you can read it again.

You can also create more than one data list. If for example you required not only a data list of lottery numbers, you also required a list of lottery numbers on your lottery ticket then you will require two separate data lists. You create the data as follows:

```
lotterydata:
DATA 12,23,34,45,56,67
DATA 23,34,45,56,67,11
DATA 34,45,56,67,33,22

ticketdata:
DATA 01,02,03,04,05,06
DATA 21,32,43,24,13,22
```

To print the first set of data to the screen, you would first point the data pointer to the first set of data. You do this by using the RESTORE command and specifying the label that precedes the data statements. A label statement is declared by adding a colon (:) as the last character of the label. You can point the data as follows:

```
RESTORE lotterydata
READ a,b,c,d,e,f
PRINT "LOTTERY ",a,b,c,d,e,f
```

Then when you wish to print out the first ticket number from the second data list, you simply use the second label that points to the ticket data:

```
RESTORE ticketdata
READ a,b,c,d,e,f
PRINT "TICKET ",a,b,c,d,e,f
```

There are better ways to structure the reading of data from a data list, but these simple examples demonstrate how to access multiple lists of data.

BRANCH STATEMENTS

Normally, a program executes statements in sequence starting at the top. A branch statement allows you to jump to another part of the program to continue execution. A GOSUB command will jump to a label and continue from its new location. When the program encounters a RETURN command, the program will jump back to the GOSUB from where it originally came. Take the following example:

```
PRINT "Hello"
GOSUB MySubroutine
END

MySubroutine:
PRINT "World"
RETURN
```

The program will print the "Hello" text to the screen, then jump to the MySubroutine line of the program and continue execution. The next command it finds will print "World" to the screen. The RETURN command then returns the program to the point it left, where it then proceeds onto the next command after the GOSUB command which in this case is the END command.

A GOTO command however, does not remember from where it jumped and will continue running from its new location permanently. It is not recommended you use GOTO commands often, as there are better ways to control the flow of your programs. Here is an example, however, of a simple GOTO command:

```
MyLabel:
PRINT "Hello World ";
GOTO MyLabel
```

Or alternatively:

```
DO
PRINT "Hello World ";
LOOP
```

You will agree the last example is a much better, cleaner and friendly way of doing the above and demonstrates how the use of GOTO can be eliminated. GOTO is retained in the Dark Basic Professional language for compatibility with older BASIC languages.

FOR NEXT Statements

You may recall the use of the FOR NEXT statement in earlier examples. The FOR NEXT commands are used to create a finite loop in which a variable is incremented or decremented from a value to a value. A simple example would be:

```
FOR T=1 TO 5
PRINT T;" ";
NEXT T
PRINT "Done"
```

The output to the screen would read:

```
1 2 3 4 5
```

The program would set T to a value of 1 and then go to the next line to PRINT. After the print, the NEXT command would return the program to the FOR command and increment the value of T to make it 2. When the PRINT command is used again, the value of T has changed and a new value is printed. This continues until T has gone from 1 through to 5, then the loop ends and the program is permitted to continue. The next command after the NEXT statement prints "Done" to the screen slowing the program has left the loop.

You can also nest loops to create a loop within a loop, as the following example shows:

```
FOR A=1 TO 5
PRINT "MAIN A=";A
FOR B=1 TO 10
```

```
PRINT "LITTLE B=";B
NEXT B
NEXT A
```

The FOR NEXT statement loops the main A variable from 1 to 5, but for every loop of A the FOR NEXT statement inside the first loop must also loop its variable B from 1 to 10. This is known as a nested loop as the loop in the middle is nested inside an outer loop.

Such loops are especially useful for working on array data by using the variables that increment as position indexes for the arrays. As an example, we could list all our lottery numbers using the following example:

```
FOR week=1 TO 52 STEP 4
PRINT "LOTTERY NUMBER FOR WEEK ";week; " ARE ";
FOR index=1 to 6
PRINT lottery(week,index);" ";
NEXT index
NEXT week
```

Notice the new STEP command added to the end of the FOR NEXT statement. The STEP command is used to change the default increment value from 1 to another value. In this case, the program will only print the lottery numbers for every forth week.

IF THEN Statements

The IF statement allows your program to make decisions that controls the flow of your program. The IF statement requires an expression to evaluate that results are either true or false. If the expression is true, the commands following the THEN command will be executed. If the expression is false, the program will move onto the next statement and ignore the rest of the IF THEN statement. Take the following example:

```
INPUT "Enter Your Age>",age
IF age>=16 THEN PRINT "You can buy a lottery ticket"
```

When the user enters a value greater or equal to 16, the program will display its message. This program demonstrates a simple IF THEN Statement. To understand how this works we must look at the IF command in a little more detail. First, we must take the expression and evaluate it:

```
age>=16
```

We can determine from our earlier coverage of operators, that this relational operator will result in either a zero or a one depending on whether age is greater or equal to 16. The IF command considers a value of zero to be false and all other values as true. So we can determine that if age is indeed greater or equal to 16 then the result will be 1, and the expression according to the IF command will be true.

The expression can be any combination of values, variables, arrays and operators providing the expression makes sense. These expressions will make sense:

```
IF A THEN PRINT "ok"
IF A = B THEN PRINT "ok"
IF A > (B - 5) THEN PRINT "ok"
IF A = (B + (A * 2)) THEN PRINT "ok"
IF A=1 AND B=2 THEN PRINT "ok"
IF NAME$="FRED" AND SURNAME$="BLOGGS" THEN PRINT "ok"
IF A#=1.5 OR LOTTERY(10,2)=20 THEN PRINT "ok"
```

These expressions will not make sense:

```
IF A = B = THEN PRINT "not ok"
IF > A = B THEN PRINT "not ok"
IF A B THEN PRINT "not ok"
IF AND A THEN PRINT "not ok"
IF B OR THEN PRINT "not ok"
```

On occasions where one line is not enough after the THEN command, you can use the IF ENDIF statement. Using the same IF logic as above, instead of a THEN Command, simply provide your commands to be executed on the lines following the IF command. You must then mark the end of the commands to be executed with an ENDIF command, as the following example shows:

```
IF A = B
PRINT "Hello A and B!"
ENDIF
```

This is the same as:

```
IF A = B THEN PRINT "Hello A and B!"
```

But the main advantage is that the first piece of code can be adapted to do this:

```
IF A = B
PRINT "Hello A!"
PRINT "Hello B!"
PRINT "Hello A and B!"
PRINT "Hello B and A!"
PRINT "Hello Everything!"
ENDIF
```

You can also respond to an IF command if the expression turns out to be false. In cases where you wish to execute a different piece of code if the condition is false, the ELSE command should be used as follows:

```
IF A = B
PRINT "The values are the same!
ELSE
PRINT "The values are different!"
ENDIF
```

It is important to make sure that you always use an ENDIF when THEN is not in use. You will note ENDIF is used whether or not the ELSE command is utilized.

PRINT Statements

The PRINT command is capable of printing out more than a single value. The command allows you to specify a list of comma separated data. The data items can be of any type. Although the use of PRINT has been frequent in the above examples, there are some unique features you may not be aware of.

When the PRINT command is used to print data to the screen, the print cursor that is used to paste the individual letters to the screen reset to the left of the screen and one line down when the print is complete. A string of PRINT commands will print to the screen one line at a time. You can change this by leaving the cursor at the end of the printed line after a PRINT command. You achieve this by adding a semi-colon (;) at the end of the print data, for example:

```
PRINT "Hello ";
PRINT "World"
```

In the same way, you can use this symbol to separate data in the same PRINT command, for example:

```
PRINT "My name is ";name$, " and I am ";age;" years old."
```

In addition to preventing the text cursor from resetting, you can also position the text cursor anywhere on the screen using the SET CURSOR command. This example will randomly print text anywhere on the screen:

```
DO
SET CURSOR RND(640),RND(480)
PRINT "TEXT"
LOOP
```

There are much more sophisticated text commands in Dark Basic Professional that handle fonts, colors, sizes and styles but you will discover these as you explore the rest of the help system.

INPUT Statements

Though seldom used, the INPUT command is a very simple way of obtaining numeric and string input from the user. You can obtain a particular type of value simply by using a variable of that type, for example:

```
INPUT a$
```

Will accept a string input from the user. If they enter a number, it will be stored as a string in the variable A$. Your program can provide a prompt to make sense of the requested input, as follows:

```
INPUT "What is your password? ",password$
```

This example prompts the user by printing the message to the screen and the user can enter the password. They will see their entry as they type it into the keyboard, and this entry will be stored in the string variable when they hit the return key. The same applies to integer and real variables.

END and BREAK Statements

The END command will terminate the execution of a program. If you were running the program from the editor, you will be dropped into the Command Line Interface (CLI). By using the END command, the user will not be able to continue running the program after they have terminated. If you were running the program as a standalone executable, the user will be returned to Windows.

If the BREAK command was used in the program, execution will merely be suspended, and not terminated. BREAK commands are used to temporarily break out of a program and into the CLI for debugging purposes. You are able to continue running the program after a BREAK command has occurred.

Common and User Functions

COMMON FUNCTIONS

Functions can be described as commands that return a value. Dark Basic Professional uses arithmetic functions, string functions, command specific functions and user-defined functions. They all share commonalties that will help you recognize what they look like and how they are used.

A simple arithmetic function is the ABS command, which takes a negative value and converts it to positive:

```
PRINT ABS(-100)
```

Will print 100 as the result of the function

The same function can be used in a calculation:

```
A = B + ABS(-100)
```

Or used with a variable:

```
A = ABS( B )
```

Or used as part of a conditional expression:

```
IF ABS( A ) > 180 THEN PRINT "ok"
```

Just as you have become accustomed to using variables in place of standard numbers and strings, you can use functions in the same way. As shown, functions can take data but they don't have to. Some functions merely return a value, such as:

```
DO
PRINT TIMER()
LOOP
```

You will notice that even though no parameter data is required, you still need to add the brackets. The brackets instruct Dark Basic Professional it is a function and is a handy way of quickly determining whether it's a variable or function. Unlike variable and array names, functions only require a dollar symbol ($) if a string is to be returned. You do not need to specify a hash symbol (#) if the function is to return a real number, as the individual commands help will reveal.

USER DEFINED FUNCTIONS

There will come a time when the ability to create your own functions will be priceless. Experienced programmers would not be able to write effective code without them. Although GOSUB commands and subroutines have been provided for compatibility and learning, it is expected that you will progress to use functions as soon as possible.

Functions are blocks of commands that usually perform a recursive or isolated task that is frequently used by your program. Variables and arrays used within the function are isolated from the rest of the program. If you use a variable name of FRED in your function, it will not affect another variable called FRED in your main program, nor any other function that happens to use a similar variable name. This may seem to be a restriction, but forces you to think about cutting up your program into exclusive tasks which is a very important lesson.

You can pass up to 255 parameters into your function, and have the option of returning a value when the function returns. Functions that do not return a value can also be used as normal commands in your main program.

Declaring a function couldn't be simpler. To use the FUNCTION command, simply provide it with a name and a list of parameters in brackets and your declaration is half-complete. Enter the commands you want on the following lines and then end the function declaration with the command ENDFUNCTION. The following example declares a function that returns half of a value passed in:

```
FUNCTION halfvalue(value)
value=value/2
ENDFUNCTION value
```

This declaration creates a function that can be used as a better print command:

```
REM Start of program
BetterPrint(10, 10, "Hello world")
END

FUNCTION BetterPrint(x, y, t$)
SET CURSOR x,y
PRINT t$
ENDFUNCTION
```

Reserved Words, Remarks and Spacing

RESERVED WORDS

Words that are deemed to be reserved are the commands of the language. You will not be able to name your variables, arrays or user functions if they exist as part of the language as commands. As you become more familiar with the language, you will be able to naturally avoid using reserved words for your variable and array names.

REMARKS

You are able to write statements in your program that will be completely ignored when run. This may seem strange at first, but actually provides one of the most important features of your programming armoury. Remarks, also known as code commentary or documentation, are the plain English descriptions of what your program does at any point in the code.

Although BASIC is quite concise as a readable language, it's never entirely clear what a piece of code is supposed to do if you read it for the first time. Do not be fooled into thinking the code you write yourself is for your eyes only. Returning to a piece of code you write 3 months previous will dispel any concerns you may have over the importance of remarks.

The trick is to write only a summary of what a piece of code does. Leave out details, data and snippets that may change or be removed. Avoid typing a half-page description as the likelihood is that you'll never find the time to read it. Remarks become a highly powerful part of your program if used correctly.

You can use the REM command to make a comment, as follows:

```
REM Print a greeting for the user
PRINT "Hello World"
```

There are other ways to make remarks in a program, such as the single open quote symbol:

```
` Print a greeting for the user
PRINT "Hello World"
```

If you wish to 'comment out' a number of lines in your program, you can avoid adding a REM command at the start of each line by using the following:

```
REMSTART
PRINT "Hello World"
PRINT "Hello World"
PRINT "Hello World"
REMEND
```

Anything between the REMSTART command and the REMEND command will be ignored when the program is run. These commands are especially useful for temporarily removing parts of the program for testing purposes.

USE OF SPACING

Unlike other BASIC languages, spaces are very important in your programs. It is important to separate commands and parameters, otherwise your program will not be understood. Take for example the line:

```
FOR T=1 TO 10
```

This would not be recognized if written as:

```
FORT=1TO10
```

Nor would this be easy to read either. Providing you use spaces to separate commands and parameters, you will encounter no problems with spacing.

Lists, Stacks and Queues

The unified array system is a remarkable method of managing your data with the minimum of fuss. Rather than maintaining different kinds of data storage, the unified array system combines the best features of all standard data structure management into a single method of access. The upshot of U.A.S is that you can initially start access of your data from an array, then decide to treat it as a stack, then decide to access it as a queue, and then back to an array. All this without loosing your data and without the need to manage its size or contents.

ARRAYS

Simple arrays form the basic foundation of the U.A.S system. Create an initial dynamic multidimensional array and access it through subscripts.

LISTS

Similar to an array, a list can expand and shrink based on the number of items contained. A list can theoretically have no items of data contained within. In contrast, it can store as much data as your memory availability allows. Accessing data through lists allows faster access than arrays in that there is no need to skip redundant items within the data. Items added and removed from a list are done so efficiently, removing the need for large clunky array sorting. You can also traverse a list without the need to know its size.

The list commands are:
EMPTY ARRAY
ARRAY INDEX TO BOTTOM
ARRAY INDEX TO TOP
ARRAY INSERT AT BOTTOM
ARRAY INSERT AT TOP
ARRAY INSERT AT ELEMENT
ARRAY DELETE ELEMENT
NEXT ARRAY INDEX
PREVIOUS ARRAY INDEX
ARRAY COUNT()
ARRAY INDEX VALID()

STACKS

A stack is summed up by the term 'last on, first off'. Stacks collect data added to it in a linear sequence of items, and the data is removed in the reverse order in which the items where added. Stacks are good for storing data prior to accessing a recursive function, or building time based data. It is a simple mechanism which removes the need for subscript or index control.

The stack commands are:
ARRAY INDEX TO STACK
ADD TO STACK
REMOVE FROM STACK

QUEUES

A queue is summed up by the term 'first on, first off'. Queues collect data added to it in a linear sequence of items, and the data is removed in the order in which the items where added. Queues are good for buffering data for later processing, where the order sequence of data is important.

The queue commands are:
ARRAY INDEX TO QUEUE
ADD TO QUEUE
REMOVE FROM QUEUE

Numerical Bases

Numerical bases are the method by which a value is represented. Most number systems use Base 10 which essentially consists of digits zero through nine. This however is not the only method of representing a value. There are many other forms of numerical representation, some of which are supported below:

BINARY

A numerical value representing Base 2 and consists only of values zero and one. Counting from zero through to ten, the binary values would look like 0, 01, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010. A binary value is coded with a '%' symbol prefix to inform the compiler that the value is binary. In binary, the value ten would be coded %1010.

OCTAL

A numerical value representing Base 8 and consists of values zero through seven. Counting from zero through to ten, the octal values would look like 00, 01, 02, 03, 04, 05, 06, 07, 10, 11, 12. An octal value is coded with a '0c' prefix to inform the compiler that the value is octal. In octal, the value ten would be coded 0c12.

HEXADECIMAL

A numerical value representing Base 16 and consists of values 0 through F. Counting from zero through to twenty, the hexadecimal values would look like 00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 0A, 0B, 0C, 0D, 0E, 0F, 10, 11, 12, 13, 14. A hexadecimal value is coded with a '0x' prefix to inform the compiler that the value is hex. In hexadecimal, the value twenty would be coded 0x14.

Errors and Warnings

There are many types of errors you will encounter while programming and running your creations, but they fall into four main categories:

'SYNTAX ERROR'

Means you have tried to compile your program but a command statement was not recognized or understood.

'PARAMETER MISMATCH'

Means you have tried to compile your program but a known command has been given the wrong parameter data. You must provide the correct types of parameters for the command, and in the right order.

'RUNTIME WARNING'

Means the program has compiled and run successfully, but the command failed due to the reasons given with the warning. A warning will not be given in a standalone executable version of the program!

'RUNTIME ERROR'

Means the program has compiled and run successfully, but the command failed due to the reasons given with the error. An error will terminate a standalone executable version of the program and provide a reason for the error!

TUTORIALS

Use these tutorials in conjunction with the source code supplied with DarkBASIC Professional. To save you typing in the texts below, just load the whole game and study each module of the source code as you read through the tutorial. Once you're done, we hope you'll take the game further and design your own version.

The source code to these tutorials can be found installed in this path:

C:/Program Files/Dark Basic Software/Dark Basic Professional/Help/Tutorials

Making A Simple FPS Game

The following tutorials will walk you through the steps involved in creating a simple 'First Person Shooter' game, otherwise known as an FPS game. There are many types of game genres you can produce, and all share commonalities with each other. As you work through these tutorials, you will learn some of the features all games share such as a game loop, player handling, enemy control and most importantly game logic.

When you have completed the tutorials, and your game is finished, the fun need not stop. If you have one of the popular Quake games such as "Half Life ®" or "Return To Castle Wolfenstein ®", you can replace the world file from the tutorial and run around shooting your enemies in a level from your favourite game.

TUTORIAL ONE
Designing Your Game

The first step in creating a large program is understanding exactly what you need to do. There are different ways to know what this is, and one of the traditional methods is a pen and a piece of paper. You already have an idea what you want to create, and its amazing how many details surface when you apply those thoughts to paper.

The above illustration demonstrates the kind of things that surfaced for the programmer who wrote these tutorials when faced with the challenge of writing an FPS Game.

TUTORIAL TWO
Collecting Your Media

The next step is locating and preparing all your media. Media can be any file that adds into your main program, usually in the form of 3D worlds, 3D models, sounds, music, images and pictures. We will require most of these types of media to create our FPS Game. Though you would resource these media files yourself from Model, Audio and Resource collections, we have prepared the media you will require for this tutorial.

The media includes a BSP World file for the game level. A sky sphere model for the sky backdrop. A gun model for the player view. An enemy model for the foe we must vanquish. A selection of sounds to bring the game to life including a large looping wave file for our in-game music. A fire image for our particle bullet and a few miscellaneous items to make the game look nicer.

All these media files were sourced from the Dark Basic Professional Resources collection and are all royalty free. It is important that all the media you use is likewise copyrighted to owners who have given express permission before you use their work. The media files once collected are arranged in suitable folders and renamed for easy reference when we focus on our program and away from the names of all our media files.

TUTORIAL THREE
The Program Skeleton

The skeleton of the program is the first thing you will be thinking about after the design stage. Most game skeletons are very similar and include the following. A setup stage to load and prepare your objects. A main loop to run your game logic and a cleanup stage where all the loaded media is released.

In Dark Basic Professional you only have to concern yourself with the first two in most cases. All media loaded is released automatically when you leave a Dark Basic Professional program. The first and second stages are very easy to visualise and can be coded in the following two steps. You must first create your setup stage:

```
rem TUT3A
rem Initial settings
sync on : sync rate 100
backdrop off : hide mouse

rem Load all media for game
gosub _load_game

rem Setup all objects for game
gosub _setup_game
```

You then create the main loop and any calls you will make over and over again to control the elements within your game:

```
rem TUT3B
rem Game loop
do

 rem Control game elements
 gosub _control_player
 gosub _control_gunandbullet
 gosub _control_enemies

 rem Update screen
 sync

rem End loop
loop
```

The last step is to create all the subroutines that the first two stages call. Subroutines are blocks of code you can jump to in order to execute instructions that achieve a subtask. A subtask could be something like loading the player, controlling the player, deleting the player and so on. It is easier to understand what a subroutine does when given a useful name:

```
rem TUT3C
_control_player:
return

_control_gunandbullet:
return
```

```
_control_enemies:
return

_control_stats:
return

_setup_game:
return

_load_game:
return
```

As you can see, each subroutine has a name that describes what it does. At the moment they are all empty and contain no code. Later on in the tutorial we shall fill these subroutines with commands.

TUTORIAL FOUR
Loading The World

The first order of business is to create a world in which our game will be set. Once upon a time this was a very difficult proposal and would often become the first major stumbling block for many would be game writers. With the built in BSP commands however, such a task is mind bogglingly simple.

With a single command we can load an entire BSP world. The BSP file contains information about polygons, collision, textures and even special effects also known as shaders. With a second command we can create a skysphere to provide an enclosed environment for our world:

```
rem TUT4A
rem Load BSP world and sky model
load bsp "world/ikzdm1.pk3","ikzdm1.bsp"
SkyObj=1 : load object "models/sky/am.x",SkyObj
```

When the BSP world and sky are loaded, we must setup the camera and ensure the autocam does not start moving around without our direct control. We must also scale down the size of the sky to bring it closer so we bring it within the maximum distance of our 3D universe and switch off sky culling so polygons that would normally show from the outside of the sky sphere will now show on the inside as well:

```
rem TUT4B
rem Setup camera
set camera range 1,10000
autocam off

rem Setup sky model
set object SkyObj,1,0,0,0,0,0,0
scale object SkyObj,20,20,20
```

In order that the user stay informed, we will provide a text prompt when the BSP is loading. Sometimes BSP worlds can be very large and can take minutes to fully load, so an onscreen prompt is very important. In order to make our game as presentable as possible, we will choose a nice large font:

```
rem TUT4C
rem Select font
set text font "arial" : set text size 16
set text to bold : set text transparent

rem Loading prompt
sync : center text screen width()/2,screen height()/2,"LOADING" : sync
```

The reason for the pre and post additions of the SYNC command is that we have initialised our program with the SYNC ON command. This tells the computer that we would like to update the screen ourselves; manually. In order to see the loading prompt, a sync must be made after we have drawn the text.

TUTORIAL FIVE
Adding A Player

After the world has been loaded, we must create a player for our game. As a first person shooter, the player is essentially the camera. You move the camera and are treated to a view of the world in the first person.

So that we can re-use our player initialisation code, we will place the code somewhere we can trigger it by the simple switching of a variable value. So that the player is initialised the first time we run the game, we will set the trigger variable first:

```
rem TUT5A
rem Trigger player initialisation
restart=1
```

We then provide the code that will setup the player when this variable is set to one. By using BSP collision on the camera, we completely automate the handling of collision within the 3D world. This not only provides sliding collision on any surface, but does so at great speed:

```
rem TUT5B
rem In case of restart
if restart=1
   restart=0
   set bsp collision off 1
   rotate camera 0,0,0
   position camera 2,2,2
   set bsp camera collision 1,0,0.75,0
endif
```

After the player has been setup, we can start to control it. We will first need to provide control of the players rotation, movement and gravity:

```
rem TUT5C
rem Control player direction
rotate camera camera angle x(0)+(mousemovey()/2.0),camera angle y(0)+(mousemovex()/2.0),0

rem Control player movement
cx#=camera angle x(0) : cy#=camera angle y(0)
if upkey()=1 then xrotate camera 0,0 : move camera 0,0.2 : xrotate camera 0,cx#
if downkey()=1 then xrotate camera 0,0 : move camera 0,-0.2 : xrotate camera 0,cx#
if leftkey()=1 then yrotate camera 0,cy#-90 : move camera 0.2 : yrotate camera 0,cy#
if rightkey()=1 then yrotate camera 0,cy#+90 : move camera 0.2 : yrotate camera 0,cy#
if wrapvalue(camera angle x(0))>40 and wrapvalue(camera angle x(0))<180 then xrotate camera 0,40
if wrapvalue(camera angle x(0))>180 and wrapvalue(camera angle x(0))<280 then xrotate camera 0,280

rem Apply simple gravity to player
position camera camera position x(),camera position y()-0.1,camera position z()
```

In addition, we must ensure the player is always at the center of the universe. To that end, when ever the player moves we must adjust the position of the sky and the 3D listener which represents the player in our game:

```
rem TUT5D
rem Player is always focal point of sky
position object SkyObj,camera position x(),camera position y(),camera position z()

rem Position listener at player for 3D sound
position listener camera position x(),camera position y(),camera position z()
rotate listener camera angle x(),camera angle y(),camera angle z()
```

TUTORIAL SIX
Adding A Gun

It would be a very short shooter game if our player did not have a gun, and bullets to fire. In traditional first person style, the gun protrudes from the base of the screen and the bullets fire into a crosshair fixed to the center of the screen.

The first step is to load all the models, sounds and images we will need to create the visual effect. We must load a gun model, some appropriate sounds, some sniper music and a crosshair for the screen:

```
rem TUT6A
rem Load model for gun
GunObj=2 : load object "models/gun/gun.x",GunObj

rem Load all sounds
GunSnd=1 : load sound "sounds/gun.wav",GunSnd
ImpactSnd=2 : load 3dsound "sounds/impact.wav",ImpactSnd
DieSnd=3 : load sound "sounds/die.wav",DieSnd

rem Load music (WAV best for looping)
MusicSnd=101 : load sound "sounds/ingame.wav",MusicSnd
loop sound MusicSnd : set sound volume MusicSnd,80

rem Load images
FireImg=1 : load image "images/fire.bmp",FireImg
CrossHairImg=2 : load image "images/crosshair.bmp",CrossHairImg
```

Once the media has been loaded, we can setup the objects before the game begins. The gun needs to be locked to the screen as it is part of the player camera now. We also need to create a bullet object and the crosshair for the screen:

```
rem TUT6B
rem Setup gun for player
lock object on GunObj
scale object GunObj,2,2,4
rotate object GunObj,270,0,0
position object GunObj,0.5,-1,2
disable object zdepth GunObj

rem Create object for bullet
BulletObj=3 : make object cube BulletObj,0.1

rem Create simple sprite based crosshair
sprite 1,320-16,240-16,CrossHairImg
set sprite 1,0,1
```

When the user presses the mouse button, the gun must be fired. In firing the gun we must first make sure the gun has cooled down enough to allow another shot. This is a storyline to cover for a technical limitation as our tutorial game allows only one bullet to be fired at any one time. When the gun is fired, a sound is played and the bullet is created at the players position and rotated to face the target:

```
rem TUT6C
rem Control gun firing
if mouseclick()=1 and bullet=-50
 bullet=100
 play sound GunSnd
 position object BulletObj,camera position x(0),camera position y(0),camera position z(0)
 rotate object BulletObj,camera angle x(0),camera angle y(0),0
 set bsp object collision 2,BulletObj,0.1,1
 move object BulletObj,0.2
endif
```

During the life of the bullet, we must move it forward and ensure it is destroyed when it hits a wall. We use the built in BSP collision setup in the previous code segment to make this task simple:

```
rem TUT6D
rem Control life of bullet
if bullet>0

 rem If bullet collides with BSP world
 if bsp collision hit(2)=1 or bulletimpact=1
  rem End bullet on wall
  position sound ImpactSnd,object position x(BulletObj), object position y(BulletObj), object position
z(BulletObj)
  play sound ImpactSnd
  bulletimpact=0
  bullet=0
 else
  rem Move bullet
  dec bullet
  move object BulletObj,0.5
 endif

 rem Bullet dies
 if bullet=0
  set bsp collision off 2
 endif

else
 rem Gun recharge phase
 if bullet>-50 then dec bullet
endif
```

TUTORIAL SEVEN
Adding Enemies

All shooters need something to shoot at. In this case we opt for those poor aliens again. They may have come to visit us with a message of peace, but our player is interested only in target practice.

The first step is to load the models for our enemy objects:

```
rem TUT7A
rem Load models for enemies
EneObj=11
for ene=EneObj to EneObj+4
 load object "models/enemy/H-Alien Psionic-Idle.x",ene
 append object "models/enemy/H-Alien Psionic-Die.x", ene, total object frames(ene)+1
 position object ene,2,2,4
 loop object ene,0,25
next ene
```

Not forgetting the sounds we will need to give our enemies a realistic presence within the game. With additional use of 3D sounds and a scary alien breathing sound we can create the creepy effect of 'what's just around the corner' for our game:

```
rem TUT7B
EnemySnd=11 : load 3dsound "sounds/enemy.wav",EnemySnd
EnemygunSnd=12 : load 3dsound "sounds/enemygun.wav",EnemygunSnd
EnemydieSnd=13 : load 3dsound "sounds/enemydie.wav",EnemydieSnd
```

Now we have loaded our enemy media, we must control them within our game. We must control each enemy, ensuring we can move, rotate, handle gravity and play 3D sound for them:

```
rem TUT7C
rem Variable for finding closest enemy
cdist#=9999.99
```

```
rem Handle enemies within world
for ene=EneObj to EneObj+4

 rem If enemy alive
 if object visible(ene)=1

 rem Kill this enemy
 killenemy=0

 rem Move enemy on a slow curve for appearance of intelligence
 if object angle z(ene)=0
   yrotate object ene,wrapvalue(object angle y(ene)+2)
 endif
 if object angle z(ene)=1
  yrotate object ene,wrapvalue(object angle y(ene)-2)
 endif
 if object angle z(ene)=2
  move object ene,0.05
 else
  move object ene,0.02
 endif

 rem Switch direction of curve based on a random value
 if rnd(200)=1 then zrotate object ene,rnd(1)

 rem Handle gravity for enemy
 position object ene,object position x(ene),object position y(ene)-0.01,object position z(ene)

 rem Work out angle and distance between enemy and player
 dx#=camera position x(0)-object position x(ene)
 dy#=camera position y(0)-object position y(ene)
 dz#=camera position z(0)-object position z(ene)
 dist#=abs(sqrt(abs(dx#*dx#)+abs(dy#*dy#)+abs(dz#*dz#)))
 viewa#=wrapvalue(atanfull(dx#,dz#))
 obja#=wrapvalue(object angle y(ene))
 if viewa#>180 then viewa#=viewa#-360
 if obja#>180 then obja#=obja#-360

 rem Closest enemy emits the enemy sound
 if dist#<cdist#
  cdist#=dist#
  position sound EnemySnd,object position x(ene), object position y(ene), object position z(ene)
  position sound EnemygunSnd,object position x(ene), object position y(ene), object position z(ene)
  position sound EnemydieSnd,object position x(ene), object position y(ene), object position z(ene)
 endif
 rem Hide enemy when die animation over
 if object frame(ene)>26+25 and object visible(ene)=1
  killenemy=1
 endif

 if killenemy=1
  set bsp collision off 3+(ene-EneObj)
  hide object ene : dec aliensleft
  killenemy=0
 endif

 rem If enemy alive ENDIF
 endif

next ene
```

To make sure that our aliens appear creepy, we will constantly play their presence sound effect, and scale the 3D sound listener so you only hear them when they are very close to the player:

```
rem TUT7D
rem Start the enemy presence sound
loop sound EnemySnd
scale listener 0.1
```

TUTORIAL EIGHT
Adding Logic

Logic is the cornerstone of every game, no matter what kind of game it is. Logic is responsible for everything that happens in the game. In this sense, logic is the code to control when you kill an alien, when the alien kills you, when you win the game and when you lose the game. As this tutorial does not attempt to write a complete game, we will add the logic to restart when the player dies and the logic to move the aliens around the world and allow the player to shoot them.

We can give our aliens the appearance of intelligence by letting them move towards the player when they are looking in the right direction, and the player is close enough:

```
rem TUT8A
rem If enemy 'facing player' and 'on similar height' and 'close', zoom in
if abs(viewa#-obja#)<10.0 and abs(dy#)<5.0 and dist#<30.0
 if object angle z(ene)<>2 then play sound EnemygunSnd
 rotate object ene,0,viewa#,2
 set object speed ene,2
else
 set object speed ene,1
endif
```

When the player gets too close to the enemy, the player must die. As we wish the game to continue, we shall reset the players position and in order to avoid the enemy constantly attacking the player, we remove the alien from the game:

```
rem TUT8B
rem If enemy gets too close to player, player dies
if dist#<2.0
 play sound DieSnd
 for x=0 to 100
  point camera object position x(ene), object position y(ene)+(x/20.0), object position z(ene)
  sync
 next x
 restart=1
 killenemy=1
endif
```

When the bullet gets too close to an enemy, both the enemy and the bullet die. The bullet will of course live again when the player fires the gun, but the enemy alas will not be so fortunate this game:

```
rem TUT8C
rem If enemy and bullet in same space, enemy dies
if bullet>0
 if object collision(BulletObj,ene)>0
  play sound EnemydieSnd
  play object ene,26,26+50
  set object speed ene,1
  bulletimpact=1
 endif
endif
```

Having all the enemies in front of you does not make for a very good game, so adding logic to space them out and give them collision so they don't move through walls provides good enemy logic for the game:

```
rem TUT8D
rem Place enemies throughout world and set BSP collision for them
aliensleft=0
restore EnemyPosData
for ene=EneObj to EneObj+4
 read ex#,ey#,ez#
 position object ene,ex#,ey#,ez#
 set bsp object collision 3+(ene-EneObj),ene,0.75,0
 yrotate object ene,180 : fix object pivot ene
 inc aliensleft
next ene
```

As the above code uses the RESTORE and READ commands, we must provide some DATA statements containing coordinates to place the enemies throughout the world. These coordinates were calculated by walking around with the player and writing down the coordinate of the player:

```
rem TUT8E
rem Enemy position data within level
EnemyPosData:
data -9.27,9.98,-2.78
data -16.54,-0.22,19.18
data 2.0,9.0,25.0
data -2.0,-9.0,25.0
data 2,4.0,10.0
```

TUTORIAL NINE
Adding Shine

Shine is the fine layer of detail that turns a good game into a great game. Shine is why you like the latest model of car, stereo and washing machine. It's the six coats of polish you pay for through the nose.

We could add shine forever. For this tutorial we shall add just two coats of shine. Some statistical shine and particle bullet shine. For our stats, we shall inform the user how many aliens remain in the level, when the level is complete and how far away the player is to the nearest alien. The function below does all this:

```
rem TUT9A
rem Distance to next alien
if aliensleft>0 then text 20,screen height()-40,"DISTANCE READING:"+str$(abs(cdist#*100))

rem Aliens Left Stat
s$=str$(aliensleft)+" ALIENS LEFT"
if aliensleft=0 then s$="LEVEL COMPLETE!" : stop sound EnemySnd
text 640-20-text width(s$),screen height()-40,s$
```

To add our particle bullet shine, we simply have to create and control a particles object and use the position of the bullet object as our guide. There are three areas of code we must modify. The creation, control and destruction of the bullet.

Bullet creation:

```
rem TUT9B
if particles exist(1)=1 then delete particles 1
make particles 1, FireImg, 50, 0.5
set particle emissions 1,10
set particle speed 1,0.01
```

Bullet control:

```
rem TUT9C
rem Update particle using bullet object position
set particle emissions 1,1+(bullet/10)
rotate particles 1,90-object angle x(BulletObj),object angle y(BulletObj)+180,0
position particles 1,object position x(BulletObj),object position y(BulletObj),object position z(BulletObj)
```

Bullet destruction:

```
rem TUT9D
set particle emissions 1,0
```

REFERENCE

CORE COMMAND SET

These commands provide the fundamental set of instructions required by all languages to control the basics. Loop control, conditional statements, primitive input and output commands, comment instructions and mathematics.

REM
This command allows you to document your program. The REM command indicates the start of a 'remark' in your program, and all remarks are skipped by the compiler. You can use remarks to provide a better description of what your program is doing. You will find it good practice to make remarks often, as you will gradually forget what each part of your program does as time goes by.

SYNTAX
```
REM Comments Can Be Written Here
```

REMSTART
This command allows you to document large areas of your program. The REMSTART and REMEND commands define an area of your program that will be ignored by the compiler. You can use remarks to provide a better description of what your program is doing. You will find it good practice to make remarks often, as you will gradually forget what each part of your program does as time goes by. Another use of REMSTART and REMEND is to temporarily remove command sequences from your program without having to delete them. By placing these commands around the offending commands, the compiler will ignore them and will be skipped when the program is run.

SYNTAX
```
REMSTART
```

REMEND
This command allows you to document large areas of your program. The REMSTART and REMEND commands define an area of your program that will be ignored by the compiler. You can use remarks to provide a better description of what your program is doing. You will find it good practice to make remarks often, as you will gradually forget what each part of your program does as time goes by. Another use of REMSTART and REMEND is to temporarily remove command sequences from your program without having to delete them. By placing these commands around the offending commands, the compiler will ignore them and will be skipped when the program is run.

SYNTAX
```
REMEND
```

SYNC ON
This command is used to improve the performance of demanding programs that require a consistent frame rate. This is especially true of games. By default, sync is set to off which allows the system to automatically handle screen refreshing. When SYNC ON is used, your program is responsible for handling screen refreshing. You can refresh the screen using the SYNC command. When you want the system to automatically handle screen refreshing again, you can use the SYNC OFF command. By placing the SYNC command at the end of your main program loop, all drawing and refresh tasks can occur in a single call. This dramatically increases the speed and smoothness of graphical operations, allowing your programs to run at their best.

SYNTAX
```
SYNC ON
```

SYNC OFF
This command is used to improve the performance of demanding programs that require a consistent frame rate. This is especially true of games. By default, sync is set to off which allows the system to automatically handle screen refreshing. When SYNC ON is used, your program is responsible for handling screen refreshing. You can refresh the screen using the SYNC command. When you want the system to automatically handle screen refreshing again, you can use the SYNC OFF command. By placing the SYNC command at the end of your main program loop, all drawing and refresh tasks can occur in a single call. This dramatically increases the speed and smoothness of graphical operations, allowing your programs to run at their best.

SYNTAX
```
SYNC OFF
```

SYNC
This command is used to improve the performance of demanding programs that require a consistent frame rate. This is especially true of games. By default, sync is set to off which allows the system to automatically handle screen refreshing. When SYNC ON is used, your program is responsible for handling screen refreshing. You can refresh the screen using the SYNC command. When you want the system to automatically handle screen refreshing again, you can use the SYNC OFF command. By placing the SYNC command at the end of your main program loop, all drawing and refresh tasks can occur in a single call. This dramatically increases the speed and smoothness of graphical operations, allowing your programs to run at their best. It is important to note the very first SYNC will only render the back buffer and reveal the contents of that buffer on the second SYNC command, as the system is based on a double buffered refresh.

SYNTAX
```
SYNC
```

SYNC RATE
This command is used to change the default refresh rate SYNC ON uses to control the speed of the SYNC update speed. The default rate sustains the 'Frames Per Second' at no more than 40fps. You can specify an integer value from 1 to 1000 to set a new maximum rate. A rate of zero will allow the program to refresh as fast as the system will allow. A forced sync rate cannot produce an accurate rating as in order to keep the refresh smooth, the code can only work with milliseconds which is not an accurate method of timing. The system is designed to treat smoothness more important than frame rating accuracy.

SYNTAX
```
SYNC RATE Rate
```

FASTSYNC
This command will perform a regular SYNC command, and will skip processing a mandatory check for windows messages. You can use this command to squeeze a few extra clock cycles out of your main loop and make your applications faster.

SYNTAX
```
FASTSYNC
```

CLS
This command will clear the screen using the current background ink color previously specified with the INK command. This command differs from the CLS found in the Basic2D command set which will paint the screen with a specified color.

SYNTAX
```
CLS
```

SET CURSOR
This command will set the cursor position used by the PRINT command. You can use this command to place basic text anywhere on the screen. The coordinates should be integer values.

SYNTAX
```
SET CURSOR X,Y
```

PRINT
This command will print text, numbers, variables and strings to the screen. You can position where the text will print using the SET CURSOR command. You can separate items you wish to print on the same line by using either a semi-colon or a comma. If you add a semi-colon at the end of your print list, the next PRINT command will add to the end of the last print line.

SYNTAX
```
PRINT Print Statements
```

INPUT
This command will accept input data from the keyboard and store the entry in the specified variable. The string that heads the input command is optional, and allows the user to provide an on-screen prompt for the data. The data is output to the screen as it is entered.

SYNTAX

```
INPUT Print Statements, Input Variable
```

DIM
This command will create an array in which you can store your program data. You are able to create arrays with up to five dimensions. Arrays are best visualized as a line of boxes. Within each of these boxes are even smaller boxes. Contained within each of these smaller boxes are single items of data. To locate the data contained in one of these boxes, you must indicate exactly which box you wish to read from or write to. Each box has what is called an index number. Starting with the largest boxes, you provide an index number that takes you inside a box. You can then provide another index number that chooses one of the smaller boxes within the larger box. Every time you enter a new box, you are in fact indexing a new dimension in the array. Using this method of indexing the array elements, you can read and write to any value within the array. You can store only one type of data per array. Data types include integer numbers, real numbers and strings.

SYNTAX

```
DIM ArrayName(Array Dimensions)
```

UNDIM
This command will delete the specified array from memory. You must have previously created the array using the DIM command in order for UNDIM to succeed. Deleting unused arrays increases system performance.

SYNTAX

```
UNDIM ArrayName(0)
```

DO
These commands will define a loop that will repeat indefinitely. You are able to break from the loop at any time by using the EXIT command.

SYNTAX

```
DO
```

EXIT
This command allows you to break from a program loop at any time. Only control loops that have an uncertain exit condition can use this command such as DO LOOP, WHILE and REPEAT loops. EXIT will have no effect on GOTO loops during the running of your program.

SYNTAX

```
EXIT
```

LOOP
These commands will define a loop that will repeat indefinitely. You are able to break from the loop at any time by using the EXIT command.

SYNTAX

```
LOOP
```

END
This command will stop the program running. If your program was run from the editor, you will be taken to the CLI. If your program was executed as a standalone executable, you will exit your program and return to Windows.

SYNTAX

```
END
```

IF
This command will allow your program to perform a sequences of commands based on the result of a condition. If the condition is true, the commands after the THEN keyword are performed. Ensure that your commands after the THEN keyword are separated by a colon(:). If the condition is false, the rest of the line is skipped.

SYNTAX

```
IF Condition
```

ELSE
This command will allow your program to perform two different sequences of commands based on the result of a condition. If the condition is true, the commands immediately following the IF command are performed. If the condition is false, the commands immediately following the ELSE command are performed. You must always use an ENDIF to mark the end of the ELSE command sequence.

SYNTAX

```
ELSE
```

ENDIF
This command will allow your program to perform a sequences of commands based on the result of a condition. If the condition is true, the commands immediately following the IF command are performed. If the condition is false, all commands before the ENDIF command are skipped.

SYNTAX

```
ENDIF
```

REPEAT
These commands will define a loop that will repeat until the Condition becomes true. Unlike the WHILE command, the repeat loop allows the command sequence within your loop to run at least once before the Condition is checked. You are able to break from the loop at any time by using the EXIT command.

SYNTAX

```
REPEAT
```

UNTIL
These commands will define a loop that will repeat until the Condition becomes true. Unlike the WHILE command, the repeat loop allows the command sequence within your loop to run at least once before the Condition is checked. You are able to break from the loop at any time by using the EXIT command.

SYNTAX

```
UNTIL Condition
```

WHILE
These commands will define a loop that will repeat while the Condition is true. Unlike the REPEAT command, the while command ensures the Condition is true before entering the loop. You are able to break from the loop at any time by using the EXIT command.

SYNTAX

```
WHILE Condition
```

ENDWHILE
These commands will define a loop that will repeat while the Condition is true. Unlike the REPEAT command, the while command ensures the Condition is true before entering the loop. You are able to break from the loop at any time by using the EXIT command.

SYNTAX

```
ENDWHILE
```

FOR
This command will define a program loop that will loop a finite number of times. The FOR command requires a variable and two values to begin the loop. The variable stores the first value, and is incremented each loop until it reaches the second value. The size of the increment is determined by the STEP value. The NEXT command is placed to mark the end of the loop. As the variable increments during the loop, you can use its value in many ways. Optionally, you can count in descending order by specifying a high first number, a low second number and a negative step value.

SYNTAX

```
FOR Variable=Start-Value TO End-Value STEP Step-Value
```

STEP
This command will define a program loop that will loop a finite number of times. The FOR command requires a variable and two values to begin the loop. The variable stores the first value, and is incremented each loop until it reaches the second value. The size of the increment is determined by the STEP value. The NEXT command is placed to mark the end of the loop. As the variable increments during the loop, you can use its value in many ways. Optionally, you can count in descending order by specifying a high first number, a low second number and a negative step value.

SYNTAX

```
FOR Variable=Start-Value TO End-Value STEP Step-Value
```

NEXT
This command will define a program loop that will loop a finite number of times. The FOR command requires a variable and two values to begin the loop. The variable stores the first value, and is incremented by 1 each loop until it reaches the second value. The NEXT command is placed to mark the end of the loop. As the variable increments during the loop, you can use its value in many ways. You can increase the size of this increment by using the STEP command.

SYNTAX

```
NEXT Variable
```

GOTO
This command will jump to the specified label in the program. Unlike the GOSUB command, you cannot return from a call when using the GOTO command. The label can be made up from any combination of alphabetic characters, but you must end the declaration of the label using a colon(:). You only need to use a colon when you are declaring the label, and should not be used when calling the label from a GOTO command.

SYNTAX

```
GOTO JumpLabel
```

GOSUB
This command will jump to the specified label in the program. Unlike the GOTO command, you can use the RETURN command to return to the program location of the last jump performed. In order to use the GOSUB command you must create a subroutine in your program that terminates with the RETURN command. To create a subroutine you must write a label in your program followed by a sequence of commands. The label can be made up from any combination of alphabetic characters, but you must end the declaration of the label using a colon(:). You only need to use a colon when you are declaring the label, and should not be used when calling the label from a GOSUB command.

SYNTAX

```
GOSUB SubroutineLabel
```

RETURN
You can use the RETURN command to return to the program location of the last jump performed. In order to use the GOSUB command you must create a subroutine in your program that terminates with the RETURN command. To create a subroutine you must write a label in your program followed by a sequence of commands. The label can be made up from any combination of alphabetic characters, but you must end the declaration of the label using a colon(:). You only need to use a colon when you are declaring the label, and should not be used when calling the label from a GOSUB command.

SYNTAX

```
RETURN
```

SELECT
Use this command in combination with CASE, ENDCASE and ENDSELECT to create a select statement. A select statement allows you to branch to different actions based on the value of the provided variable. The variable can be an integer, real or string value.

SYNTAX

```
SELECT Variable
```

CASE
Use this command in combination with SELECT, ENDCASE and ENDSELECT to create a select statement. A case statement specifies the value that if matching the contents of the variable, will run the code below it.

SYNTAX

```
CASE Data
```

CASE DEFAULT
Use this command in combination with SELECT, ENDCASE and ENDSELECT to create a select statement. A case default statement holds code below it in the event that no case statement catches the value of the variable. The case default statement must go at the end of a sequence of case statement, but above the ENDSELECT statement.

SYNTAX

```
CASE DEFAULT:
```

ENDCASE
Use this command in combination with SELECT, CASE and ENDSELECT to create a select statement. A caseend statement specifies the end of a piece of code you are specifying for a case statement.

SYNTAX

```
ENDCASE
```

ENDSELECT
Use this command in combination with SELECT, CASE and ENDCASE to create a select statement. An endselect statement specifies the end of a select statement.

SYNTAX

```
ENDSELECT
```

FUNCTION
These commands will declare a user defined function within your program. User functions work in the same way normal commands work. They can accept multiple parameters and return values in the same manner, allowing you to create customized commands within your program.

SYNTAX

```
FUNCTION FunctionName(Function Parameters)
```

EXITFUNCTION
This command will immediately exit the current function, optionally returning a value. The remaining code between the EXITFUNCTION and the ENDFUNCTION commands will be ignored.

SYNTAX

```
EXITFUNCTION Variable
```

ENDFUNCTION
These commands will declare a user defined function within your program. User functions work in the same way normal commands work. They can accept multiple parameters and return values in the same manner, allowing you to create customized commands within your program.

SYNTAX

```
ENDFUNCTION Variable
```

RESTORE
This command will reset the DATA statement pointer to the very first piece of data in your program. Optionally, if you specify a label, it will reset to the DATA statement immediately following that label. To create a DATA statement label you must write a label in your program followed by a sequence of DATA commands. The label can be made up from any combination of alphabetic characters, but you must end the declaration of the label using a colon(:). You only need to use a colon when you are declaring the label, and a colon should not be used when calling the RESTORE command.

SYNTAX

```
RESTORE DataLabel
```

READ
This command will read next available item from the DATA statement in your program and place the resulting data in a specified variable. You can create DATA statements using the DATA command, and you can reset the pointer to the data using the RESTORE command. You can read the data as integer numbers, real numbers or strings but you must ensure the DATA statements data types are in the same order as your program reads them. If you read data into a variable of the wrong type, a zero value or empty string is written into the variable.

SYNTAX

```
READ Variable
```

DATA
This command allows you to create a sequence of data values as part of your program. The values can be integer, real or string values. You may place the data values in any order you wish providing you separate each one with a comma or place new data values in a separate DATA statement. Data values can be read from the DATA statement using the READ command, and the point at which the data is read can be reset using the RESTORE command. You can separate batches of DATA statements using labels in order to segment your data for alternative uses.

SYNTAX

```
DATA Data Statements
```

SAVE ARRAY
This command will save the contents of the array to the specified file. It is highly recommended that the array used to save the data should be identical to the one used when reloading. When specifying the array name, the array index numbers are ignored, but you must still provide them to ensure the array is recognized.

SYNTAX

```
SAVE ARRAY Filename, Array Name(0)
```

LOAD ARRAY
This command will load the contents of the file into the specified array. It is highly recommended that the array used to save the data should be identical to the one used when reloading. When specifying the array name, the array index numbers are ignored, but you must still provide them to ensure the array is recognized.

SYNTAX

```
LOAD ARRAY Filename, Array Name(0)
```

WAIT
This command will pause the program for the specified Duration. The Duration is specified in milliseconds, where 1000 units represent 1 second.

SYNTAX

```
WAIT Delay
```

SLEEP
This command will pause the program for the specified Duration. The Duration is specified in milliseconds, where 1000 units represent 1 second.

SYNTAX

```
SLEEP Delay
```

WAIT KEY
This command will pause the program from running until a key is pressed. To detect for a specific key, please refer to the INKEY$ or SCANCODE commands found in the Input Command Set.

SYNTAX

```
WAIT KEY
```

WAIT MOUSE

This command will pause the program from running until a mouse button is pressed. To detect for a specific mouse button, please refer to the MOUSECLICK() command.

SYNTAX

```
WAIT MOUSE
```

SUSPEND FOR KEY
This command will pause the program from running until a key is pressed. To detect for a specific key, please refer to the INKEY$ or SCANCODE commands found in the Input Command Set.

SYNTAX

```
SUSPEND FOR KEY
```

SUSPEND FOR MOUSE
This command will pause the program from running until a mouse button is pressed. To detect for a specific mouse button, please refer to the MOUSECLICK() command.

SYNTAX

```
SUSPEND FOR MOUSE
```

BREAK
This command will break from the program when running in Debug Mode. You can optionally specify a string as a parameter to this command to carry text to the CLI allowing you to report debug information on the break. When you break from a program, you can resume execution from the Debugger.

SYNTAX

```
BREAK
BREAK Debug String
```

DRAW TO BACK
This command will ensure all 2D activities such as drawing text, images and bitmaps are performed before any 3D is rendered. This allows you to create 2D effects in the background. In most cases 3D will obscure most if not all 2D activities and so this is not the default behaviour.

SYNTAX

```
DRAW TO BACK
```

DRAW TO FRONT
This command will ensure all 2D activities such as drawing text, images and bitmaps are performed after the 3D has rendered, allowing you to overlap the screen with 2D content. This is the default behaviour.

SYNTAX

```
DRAW TO FRONT
```

DRAW SPRITES FIRST
This command changes the order in which sprites are drawn to the screen. By calling this command, all sprites will be drawn to the screen 'before' any 3D is drawn. This will cause the sprites to appear to be behind 3D objects, and most likely obscured by them.

SYNTAX

```
DRAW SPRITES FIRST
```

DRAW SPRITES LAST
This command change the order in which sprites are drawn to the screen. By calling this command, all sprites will be drawn to the screen 'after' any 3D is drawn. This will cause the sprites to appear over the top of any 3D being rendered. This is the default behaviour.

SYNTAX

```
DRAW SPRITES LAST
```

RANDOMIZE
The RANDOMIZE command reseeds the random number generator. If the random number generator is not reseeded the RND() command can return the same sequence of random numbers. To change the sequence of random number every time the program is run, place a randomize statement with an integer number at the beginning of the program and change the value with each run.

SYNTAX
```
RANDOMIZE Seed Value
```

INC
The INC command will increment a variable by a specified Value or by a default of 1. Both integer and real variables can be incremented by either an integer or real value, in addition to array and data type elements.

SYNTAX
```
INC Variable, Value
```

DEC
The DEC command will decrement a variable by a specified Value or by a default of 1. Both integer and real variables can be decremented by either an integer or real value, in addition to array and data type elements.

SYNTAX
```
DEC Variable, Value
```

MAKE MEMORY
This command will create an area of memory of the given size in bytes. You can use dynamically created areas of memory to store any type of data using the indirection symbol. Simply place a * before the variable holding the address of the memory block to write into the first byte position. Adding to the value representing the address will move you through the memory block allowing both reading and writing to the memory. This command will return the address of the first byte of this newly created memory block.

SYNTAX
```
Return DWORD=MAKE MEMORY(Size In Bytes)
```

DELETE MEMORY
This command will delete an area of memory previously created with the MAKE MEMORY command. You must specify the first byte of the memory block to release the memory, or this command will not only fail but likely crash the computer. Use with caution.

SYNTAX
```
DELETE MEMORY Memory Address
```

FILL MEMORY
This command will fill an area of memory with a specified byte value. You can use this command in combination with the MAKE MEMORY command to clear previously created data. With care, you can also use this command to create wipe effects, clear the screen and initialise data structures. The size to clear is specified in bytes.

SYNTAX
```
FILL MEMORY Memory Address, FillByte, Size In Bytes
```

COPY MEMORY
This command will copy a chunk of data from one part of memory to another. Ideally this command is used to copy data from one memory block to another, or to and from a locked resource in memory. Look at the MAKE MEMORY command for details on how to create a memory block.

SYNTAX
```
COPY MEMORY Memory Destination, Memory Source, Size In Bytes
```

EMPTY ARRAY
This command will empty the contents of any array. The array will read as having absolutely no items, and an array size of zero.

SYNTAX
```
EMPTY ARRAY Array Name(0)
```

ARRAY INSERT AT TOP
This command will insert a blank item at the start of the array list. When a blank item is inserted into a list, all the elements thereafter are shuffled one space down to make room for the new item. Lists are an expandable and shrinkable form of array, and do not use a fixed size.

SYNTAX
```
ARRAY INSERT AT TOP Array Name(0)
ARRAY INSERT AT TOP Array Name(0), Index
```

ARRAY INSERT AT BOTTOM
This command will insert a blank item at the end of the array list. Lists are an expandable and shrinkable form of array, and do not use a fixed size.

SYNTAX
```
ARRAY INSERT AT BOTTOM Array Name(0)
ARRAY INSERT AT BOTTOM Array Name(0), Index
```

ARRAY INSERT AT ELEMENT
This command will insert a blank item at the position specified in the array list. When a blank item is inserted into a list, all the elements thereafter are shuffled one space down to make room for the new item. Lists are an expandable and shrinkable form of array.

SYNTAX
```
ARRAY INSERT AT ELEMENT Array Name(0), Index
```

ARRAY DELETE ELEMENT
This command will delete the specified item from the array. You specify the item using the array index. When the item is deleted, all elements thereafter are shuffled back one space and the size of the array is reduced accordingly.

SYNTAX
```
ARRAY DELETE ELEMENT Array Name(0)
ARRAY DELETE ELEMENT Array Name(0), Index
```

NEXT ARRAY INDEX
This command will move the internal array index to the next logical item. When the index reaches the end of the array, the ARRAY INDEX VALID() command will return a zero.

SYNTAX
```
NEXT ARRAY INDEX Array Name(0)
```

PREVIOUS ARRAY INDEX
This command will move the internal array index to the previous logical item. When the index reaches the start of the array, the ARRAY INDEX VALID() command will return a zero.

SYNTAX
```
PREVIOUS ARRAY INDEX Array Name(0)
```

ARRAY INDEX TO TOP
This command will set the array index to the start of the array. The start index of any array is always zero.

SYNTAX

```
ARRAY INDEX TO TOP Array Name(0)
```

ARRAY INDEX TO BOTTOM
This command will set the array index to the end of the array. As the size of the array can change dynamically, this command will first determine the size of the array, and then place the index at the last item.

SYNTAX

```
ARRAY INDEX TO BOTTOM Array Name(0)
```

ARRAY INDEX TO QUEUE
This command will set the current array index to the first item in the array queue. Queues are a first in, first out data structure. You can imagine a queue as a line of people waiting for the bus. The last item of data added will be the last item of data to come off the queue.

SYNTAX

```
ARRAY INDEX TO QUEUE Array Name(0)
```

ADD TO QUEUE
This command will add a blank item to the array queue. Queues are a first in, first out data structure. You can imagine a queue as a line of people waiting for the bus. The last item of data added will be the last item of data to come off the queue.

SYNTAX

```
ADD TO QUEUE Array Name(0)
```

REMOVE FROM QUEUE
This command will remove the first item of data from the array queue. Queues are a first in, first out data structure. You can imagine a queue as a line of people waiting for the bus. The last item of data added will be the last item of data to come off the queue.

SYNTAX

```
REMOVE FROM QUEUE Array Name(0)
```

ARRAY INDEX TO STACK
This command will set the array index to the end of the array stack. Stacks are a first in, last out data structure. You can imagine a stack as a pile of books. You can only take the book that was last added to the stack. To get to the item at the start of the stack, you must first remove all other items.

SYNTAX

```
ARRAY INDEX TO STACK Array Name(0)
```

ADD TO STACK
This command will add a blank item to the array stack. Stacks are a first in, last out data structure. You can imagine a stack as a pile of books. You can only take the book that was last added to the stack. To get to the item at the start of the stack, you must first remove all other items.

SYNTAX

```
ADD TO STACK Array Name(0)
```

REMOVE FROM STACK
This command will remove an item from the end of the array stack. Stacks are a first in, last out data structure. You can imagine a stack as a pile of books. You can only take the book that was last added to the stack. To get to the item at the start of the stack, you must first remove all other items.

SYNTAX

```
REMOVE FROM STACK Array Name(0)
```

COS

This command will return the cosine of value were the value is in degrees between 0 and 360. The value can be a real or integer number. The return value is a real number.

SYNTAX

Return Float=COS(Value)

SIN
This command will return the sine of value where value is in degrees between 0 and 360. The value can be a real or integer number. The return value is a real number.

SYNTAX

Return Float=SIN(Value)

TAN
This command will returns the tangent of the value. The value can be a real or integer number. The return value is a real number.

SYNTAX

Return Float=TAN(Value)

ACOS
This command will return the arccosine of a value. The value can be a real or integer number. The return value is a real number.

SYNTAX

Return Float=ACOS(Value)

ASIN
This command will return the arcsine of value. The value can be a real or integer number. The return value is a real number.

SYNTAX

Return Float=ASIN(Value)

ATAN
This command will return the tangent in degrees between 0 and 360. The value can be a real or integer number. The return value should be a real number.

SYNTAX

Return Float=ATAN(Value)

ATANFULL
This command will return the angle of two points in degrees between 0 and 360. The distance values are calculated by finding the distance between two points for both the X and Y axis. The distance values can be a real or integer number. The return value is a real number.

SYNTAX

Return Float=ATANFULL(Distance X, Distance Y)

HCOS
This command will return the hyperbolic cosine of the value. The value can be a real or integer number. The return value is a real number.

SYNTAX

Return Float=HCOS(Value)

HSIN
This command will return the hyperbolic sine of the value. The value can be a real or integer number. The return value is a real number.

SYNTAX

Return Float=HSIN(Value)

HTAN
This command will return the hyperbolic tangent of value. The value can be a real or integer number. The return value is a real number.

SYNTAX

Return Float=HTAN(Value)

SQRT
This command will return the square root of a value. The value can be an integer or real number. The return value will be a real number.

SYNTAX

Return Float=SQRT(Value)

ABS
This command will return the positive equivalent of a value. The value can be a real or integer number. The return value can be a real or integer number.

SYNTAX

Return Float=ABS(Value)

INT
This command will return the largest integer before the decimal point of a real number. The return value will be an integer number.

SYNTAX

Return Integer=INT(Value)

EXP
This command will return a result raised to the power of the input value. The value should be an integer number. The return value will be a float value.

SYNTAX

Return Float=EXP(Value)

RND
This command will return a random number between 0 and the range provided. The integer range will be the highest number you want returned. The return value will be an integer number.

SYNTAX

Return Float=RND(Range Value)

TIMER
This command will get the internal system time, which continually increments at a thousand times a second. The system time is returned in milliseconds, where 1000 units represent 1 second.

SYNTAX

Return Integer=TIMER()

INKEY$
This command will get the character string of the key currently being pressed.

SYNTAX

Return String=INKEY$()

CL$
This command will return the string passed in as a command line when the program is run as a standalone executable. When this program is created as an executable file, any additional parameters entered after the executable name are treated as a single command line string. You can use this string to control how your executable behaves based on the contents of the string.

SYNTAX
Return String=CL$()

GET DATE$
This command will return the current clock date as a formatted string. The string contains the date in the format MM/DD/YY. MM indicating the current month number 01-12, DD indicating the current date number 01-31 and YY indicating the current year number 00-99.

SYNTAX
Return String=GET DATE$()

GET TIME$
This command will return the current clock time as a formatted string. The string contains the time in the format HH:MM:SS. HH indicating the current hour number 00-23, MM indicating the current minute number 00-59 and SS indicating the current second number 00-59.

SYNTAX
Return String=GET TIME$()

CURVEVALUE
This command will return an auto-interpolated value based on a given speed. This command will gradually move a number from its current value to a destination value at a certain speed. This command can be used to create the technique of controlling a cars velocity as it breaks to a halt. The parameters should be specified using real numbers.

SYNTAX
Return Float=CURVEVALUE(Destination Value, Current Value, Speed Value)

WRAPVALUE
This command will return a value that does not exceed the range of 0 to 360. Where a value is specified that exceeds this range, the command will wrap the value around to bring it back within the range. This command is best understood by using the number of a clock as a mental picture of how a number wraps. If the clock hand points to 11 and is then advanced 2 hours, the number has to wrap from 12 around to 1 to keep the cycle going. The parameter should be specified using a real number.

SYNTAX
Return Float=WRAPVALUE(Angle Value)

NEWXVALUE
This command will return a value that represents the new X position of a point in 3D space. This command is used in conjunction with NEWYVALUE and NEWZVALUE commands to move from one point in space to another point in space based on a specified angle. Rather than using COS/SIN maths, these commands simplify the task of moving coordinates within 3D space. The step value specifies how far in the specified direction you would like to calculate. The parameters should be specified using real numbers.

SYNTAX
Return Float=NEWXVALUE(Current X Value, Angle Value, Step Value)

NEWYVALUE
This command will return a value that represents the new Y position of a point in 3D space. This command is used in conjunction with NEWXVALUE and NEWZVALUE commands to move from one point in space to another point in space based on a specified angle. Rather than using COS/SIN maths, these commands simplify the task of moving coordinates

within 3D space. The step value specifies how far in the specified direction you would like to calculate. The parameters should be specified using real numbers.

SYNTAX

Return Float=NEWYVALUE(Current Y Value, Angle Value, Step Value)

NEWZVALUE

This command will return a value that represents the new Z position of a point in 3D space. This command is used in conjunction with NEWXVALUE and NEWYVALUE commands to move from one point in space to another point in space based on a specified angle. Rather than using COS/SIN maths, these commands simplify the task of moving coordinates within 3D space. The step value specifies how far in the specified direction you would like to calculate. The parameters should be specified using real numbers.

SYNTAX

Return Float=NEWZVALUE(Current Z Value, Angle Value, Step Value)

CURVEANGLE

This command will return an auto-interpolated angle based on a given speed. This command will gradually move a number from its current value to a destination value at a certain speed. This command can be used to create the technique of a camera swinging into position from another location by curving the value of the camera angles. The parameters should be specified using real numbers.

SYNTAX

Return Float=CURVEANGLE(Destination Value, Current Value, Speed Value)

ARRAY COUNT

This command will return the number of items in the array. This is usually the value used to create the array when the DIM command was used, however by modifying the dynamic array this value can change throughout the execution of the program.

SYNTAX

Return Integer=ARRAY COUNT(Array Name(0))

ARRAY INDEX VALID

This command will return a value of one if the array index specified is a valid item within the array. Specifying a value greater or equal to the size of the array would therefore return a zero.

SYNTAX

Return Integer=ARRAY INDEX VALID(Array Name(0))

TEXT COMMAND SET

These commands provide functionality to display text within your application. You can change the text in many ways, including its color, font, style, size and position. Due to the method by which the text is displayed, they are also antialiased and very fast if supported by the hardware on your system.

TEXT
This command will output the provided string using the current text settings at the specified coordinates on the screen. The coordinates should be integer values.

SYNTAX
```
TEXT X,Y,String
```

CENTER TEXT
This command will output the provided string using the current text settings at the specified coordinates. The text will be centered on the X coordinate given. The coordinates should be integer values.

SYNTAX
```
CENTER TEXT X,Y,String
```

SET TEXT FONT
This command will set the typeface of the font you wish to use. You can optionally specify a character set value when selecting your font. The character set value allows you to use other western non-unicode languages not available through the standard ASCII character set. To switch the character set to another language, use an international charset code. These codes can be located in the Principals section along with the ASCII Character Code Lists.

SYNTAX
```
SET TEXT FONT Fontname
SET TEXT FONT Fontname, Charset Value
```

SET TEXT SIZE
This command will set the point size of any text you are about to output. The point size should be an integer value. The size of printed text will not change if the font used does not support the size specified. Using this command with SET TEXT FONT to set a font type that allows multiple sizes such as 'Arial'.

SYNTAX
```
SET TEXT SIZE Point size
```

SET TEXT OPAQUE
This command will set the background of the current text settings to the color of the background ink.

SYNTAX
```
SET TEXT OPAQUE
```

SET TEXT TRANSPARENT
This command will set the background of the text you are about to output as transparent.

SYNTAX
```
SET TEXT TRANSPARENT
```

SET TEXT TO NORMAL
This command will set text style to non-bold and non-italic.

SYNTAX
```
SET TEXT TO NORMAL
```

SET TEXT TO ITALIC
This command will set the text style to non-bold and italic.

SYNTAX
```
SET TEXT TO ITALIC
```

SET TEXT TO BOLD
This command will set the text style to bold and non-italic.

SYNTAX
```
SET TEXT TO BOLD
```

SET TEXT TO BOLDITALIC
This command will set the text style to bold and italic.

SYNTAX
```
SET TEXT TO BOLDITALIC
```

PERFORM CHECKLIST FOR FONTS
This command will build a checklist and search for all available fonts on the system. Use the CHECKLIST commands found in SYSTEM command set to access the checklist.

SYNTAX
```
PERFORM CHECKLIST FOR FONTS
```

TEXT BACKGROUND TYPE
This command will return an integer value of one if the current text settings is set to using a transparent background, otherwise zero is returned.

SYNTAX
```
Return Integer=TEXT BACKGROUND TYPE()
```

TEXT FONT$
This command will return a string containing the typeface description of the current text settings.

SYNTAX
```
Return String=TEXT FONT$()
```

TEXT SIZE
This command will return the point size of the current font described by the current text settings.

SYNTAX
```
Return Integer=TEXT SIZE()
```

TEXT STYLE
This command will return a code based on the style of the current text settings. The return value will be an integer number of zero for normal, 1 for italic, 2 for bold and 3 for bold italic.

SYNTAX
```
Return Integer=TEXT STYLE()
```

TEXT WIDTH
This command will return the width of the provided string using the current text settings.

SYNTAX

Return Integer=TEXT WIDTH(String)

TEXT HEIGHT
This command will return the height of the provided string using the current text settings.

SYNTAX
Return Integer=TEXT HEIGHT(String)

ASC
This command will return an integer value that represents the ASCII code for the first character of the provided string.

SYNTAX
Return Integer=ASC(String)

BIN$
This command will return a 32 character string equivalent to the binary representation of the specified value provided. The value provided should be an integer value.

SYNTAX
Return String=BIN$(Value)

CHR$
This command will return a character string equivalent to the ASCII character number provided. ASCII is a standard way of coding the characters used to construct strings. The value provided should be an integer value.

SYNTAX
Return String=CHR$(Value)

HEX$
This command will return an eight character string equivalent to the hexadecimal representation of the number provided. The value provided should be an integer value.

SYNTAX
Return String=HEX$(Value)

LEFT$
This command will return a string containing the leftmost characters of the string that was provided. If the value is greater than the number of characters in the string then the entire string is return. If the value is zero then an empty string is returned. The value provided should be an integer value.

SYNTAX
Return String=LEFT$(String,Value)

LEN
This command will return the number of characters in the string provided. Non-printable control characters and blanks are counted.

SYNTAX
Return Integer=LEN(String)

LOWER$
This command will return a string converting the specified string to lowercase.

SYNTAX
Return String=LOWER$(String)

MID$
This command will extract a character from the string provided. The return string will contain a single character extracted from the string specified. The value provided should be an integer value.

SYNTAX

Return String=MID$(String,Value)

RIGHT$
This command will return the rightmost set of characters. The value specifies the number of characters to extract from the string provided. The value provided should be an integer value.

SYNTAX

Return String=RIGHT$(String,Value)

STR$
This command will return a string representation of the value provided. The value provided should be an integer value.

SYNTAX

Return String=STR$(Value)

UPPER$
This command will return a string converting the specified string to uppercase.

SYNTAX

Return String=UPPER$(String)

VAL
This command will return an integer number of the string provided by converting the string to a numerical form.

SYNTAX

Return Float=VAL(String)

SPACE$
This command will produce a string containing the number of spaces specified.

SYNTAX

Return String=SPACE$(Number of Spaces)

INPUT COMMAND SET

These commands provide functionality for the control of input data into your application. Input can come from keyboard, mouse, joystick or control devices attached to your system. In addition you can control the use of force feedback devices for extra effect within your games and applications.

SHOW MOUSE
This command will let you see the mouse pointer image on screen.

SYNTAX
```
SHOW MOUSE
```

HIDE MOUSE
This command will hide the mouse pointer image.

SYNTAX
```
HIDE MOUSE
```

POSITION MOUSE
This command will change the position of the mouse pointer. By specifying a 2D screen coordinate using integer values, you can relocate the location of the mouse pointer at any time.

SYNTAX
```
POSITION MOUSE X,Y
```

CHANGE MOUSE
This command will change the cursor that belongs to the mouse pointer. A value of zero uses the applications arrow cursor and a value of one will use the hourglass cursor. Values 2 to 31 are custom cursors that can be specified in the project media section and selected with this command.

SYNTAX
```
CHANGE MOUSE Cursor Number
```

CLEAR ENTRY BUFFER
This command will clear the string current held by the windows system. This string is maintained by the windows message pump and ensures you do not miss characters typed at speed in your application. You can get the contents of this string by using the ENTRY$() command.

SYNTAX
```
CLEAR ENTRY BUFFER
```

WRITE TO CLIPBOARD
This command will write a string to the system clipboard. The system clipboard remains in tact even after the program has been terminated and can be used to transfer values from program to program, or otherwise retain string data beyond the scope of the program.

SYNTAX
```
WRITE TO CLIPBOARD String
```

WRITE TO REGISTRY
Write a value to the specified registry location. The folder name points to the general area within the registry. The key name points to the label that describes the data stored within the registry folder. The value is the integer value you wish to store.

SYNTAX
```
WRITE TO REGISTRY Folder Name, Key Name, Value
```

WRITE STRING TO REGISTRY
Write a string to the specified registry location. The folder name points to the general area within the registry. The key name points to the label that describes the data stored within the registry folder. The string is the text you wish to store.

SYNTAX

```
WRITE STRING TO REGISTRY Folder Name, Key Name, String
```

PERFORM CHECKLIST FOR CONTROL DEVICES
This command will make a checklist of all installed control devices. Control devices can range from a force-feedback joystick to head mounted displays. You can read the name of each found device by reading the Checklist String after using this command. Additionally, you can determine whether the control device has forcefeedback capability by checking for a value of 1 in Checklist Value A. In general, most devices simply provide input to the program in a standard X, Y and Z axis format. Use the CHECKLIST commands in the SYSTEM command set to read the checklist.

SYNTAX

```
PERFORM CHECKLIST FOR CONTROL DEVICES
```

SET CONTROL DEVICE
This command will select a known control device for use as the current device. You must provide the name of the device. Use the checklist command to find all the names of currently available control devices.

SYNTAX

```
SET CONTROL DEVICE Device Name$
```

FORCE UP
This command will use the current control device if it has force feedback capability. The command will force the device upward with a power of magnitude specified between 0 and 100. The magnitude should be an integer value.

SYNTAX

```
FORCE UP Magnitude Value
```

FORCE DOWN
This command will use the current control device if it has force feedback capability. The command will force the device downward with a power of magnitude specified between 0 and 100. The magnitude should be an integer value.

SYNTAX

```
FORCE DOWN Magnitude Value
```

FORCE LEFT
This command will use the current control device if it has force feedback capability. The command will force the device left with a power of magnitude specified between 0 and 100. The magnitude should be an integer value.

SYNTAX

```
FORCE LEFT Magnitude Value
```

FORCE RIGHT
This command will use the current control device if it has force feedback capability. The command will force the device right with a power of magnitude specified between 0 and 100. The magnitude should be an integer value.

SYNTAX

```
FORCE RIGHT Magnitude Value
```

FORCE ANGLE
This command will use the current control device if it has force feedback capability. The command will force the device in the direction specified by the angle value and at a power of magnitude specified between 0 and 100. The delay value specifies how many milliseconds to sustain the effect of force. The magnitude and delay should be integer values. A delay value of zero indicates an infinite effect of force. The angle value should be a real number.

SYNTAX

FORCE ANGLE Magnitude Value, Angle Value, Delay Value

FORCE NO EFFECT
This command will use the current control device if it has force feedback capability. The command will force the device to halt the effect of any forces.

SYNTAX
FORCE NO EFFECT

FORCE WATER EFFECT
This command will use the current control device if it has force feedback capability. The command will force the device to simulate the effect of moving through water. The force applied to any joystick movement will simulate a dampening effect when fast movements are attempted by the user. The magnitude value determines the strength of the force between 0 and 100. The delay value indicates how many milliseconds the effect lasts for. A delay value of zero indicates an infinite effect of force. The magnitude and delay should be integer values.

SYNTAX
FORCE WATER EFFECT Magnitude Value, Delay Value

FORCE SHOOT
This command will use the current control device if it has force feedback capability. The command will force the device to recoil like a discharging pistol at a power of magnitude specified between 0 and 100. The delay value specifies how many milliseconds to sustain the effect of force. A delay value of zero indicates an infinite effect of force. The magnitude and delay should be integer values.

SYNTAX
FORCE SHOOT Magnitude Value, Delay Value

FORCE CHAINSAW
This command will use the current control device if it has force feedback capability. The command will force the device to rattle like a chainsaw at a power of magnitude specified between 0 and 100. The delay value specifies how many milliseconds to sustain the effect of force. A delay value of zero indicates an infinite effect of force. The magnitude and delay should be integer values.

SYNTAX
FORCE CHAINSAW Magnitude Value, Delay Value

FORCE IMPACT
This command will use the current control device if it has force feedback capability. The command will force the device to shudder at a power of magnitude specified between 0 and 100. The delay value specifies how many milliseconds to sustain the effect of force. A delay value of zero indicates an infinite effect of force. The magnitude and delay should be integer values.

SYNTAX
FORCE IMPACT Magnitude Value, Delay Value

FORCE AUTO CENTER ON
This command will use the current control device if it has force feedback capability. The command will force the device to pull to a central position as though connected by springs.

SYNTAX
FORCE AUTO CENTER ON

FORCE AUTO CENTER OFF
This command will use the current control device if it has force feedback capability. The command will deactivate any effect of force that has been created with the FORCE AUTO CENTER ON command.

SYNTAX
FORCE AUTO CENTER OFF

MOUSECLICK
This command will return an integer value if a mouse button is pressed. The integer return value will depend on which mouse button has been pressed. A mouse can have up to four buttons, and each one can be detected using this command. Each button is assigned a value. The left button is assigned a value of 1. The right button is assigned a value of 2. Buttons three and four are assigned values of 4 and 8 respectively. When more than one button is pressed, the value of the buttons are added to produce a combined value you can check for.

SYNTAX
Return Integer=MOUSECLICK()

MOUSEMOVEX
This command will get the current integer X movement value of the mouse pointer. Instead of the actual mouse position, this command returns the difference between the current mouse X position and the last mouse X position.

SYNTAX
Return Integer=MOUSEMOVEX()

MOUSEMOVEY
This command will get the current integer Y movement value of the mouse pointer. Instead of the actual mouse position, this command returns the difference between the current mouse Y position and the last mouse Y position.

SYNTAX
Return Integer=MOUSEMOVEY()

MOUSEMOVEZ
This command will get the current integer Z movement value of the mouse pointer. Instead of the actual mouse position, this command returns the difference between the current mouse Z position and the last mouse Z position.

SYNTAX
Return Integer=MOUSEMOVEZ()

MOUSEX
This command will get the current integer X position of the mouse pointer.

SYNTAX
Return Integer=MOUSEX()

MOUSEY
This command will get the current integer Y position of the mouse pointer.

SYNTAX
Return Integer=MOUSEY()

MOUSEZ
This command will get the current integer Z position of the mouse pointer. The Z position usually belongs to the wheel you can sometimes find in the center of your mouse. The mouse Z position range is 0 to 100.

SYNTAX
Return Integer=MOUSEZ()

UPKEY
This command will return an integer value of one if the up arrow key is being pressed, otherwise zero is returned.

SYNTAX
Return Integer=UPKEY()

DOWNKEY
This command will return an integer value of one if the down arrow key is being pressed, otherwise zero is returned.

SYNTAX
Return Integer=DOWNKEY()

LEFTKEY
This command will return an integer value of one if the left arrow key is being pressed, otherwise zero is returned.

SYNTAX
Return Integer=LEFTKEY()

RIGHTKEY
This command will return an integer value of one if the right arrow key is being pressed, otherwise zero is returned.

SYNTAX
Return Integer=RIGHTKEY()

CONTROLKEY
This command will return an integer value of one if the control key is being pressed, otherwise zero is returned.

SYNTAX
Return Integer=CONTROLKEY()

SHIFTKEY
This command will return an integer value of one if the shift key is being pressed, otherwise zero is returned.

SYNTAX
Return Integer=SHIFTKEY()

RETURNKEY
This command will return an integer value of one if the return key is being pressed, otherwise zero is returned.

SYNTAX
Return Integer=RETURNKEY()

ESCAPEKEY
This command will return an integer value of one if the escape key is being pressed, otherwise zero is returned.

SYNTAX
Return Integer=ESCAPEKEY()

SPACEKEY
This command will return an integer value of one if the space key is being pressed, otherwise zero is returned.

SYNTAX
Return Integer=SPACEKEY()

SCANCODE
This command will get the scancode of the key currently being pressed.

SYNTAX
Return Integer=SCANCODE()

KEYSTATE
This command will return an integer value of one if the key specified by the scancode is pressed, otherwise zero will be

returned. The scancode value is the raw value assigned to the key of the keyboard device and very often is ordered sequentially from the top left of the keyboard to the bottom right.

SYNTAX

Return Integer=KEYSTATE(Scancode)

ENTRY$
This command will return the string currently held by the windows system. This string is maintained by the windows message pump and ensures you do not miss characters typed at speed in your application. You can clear this string using the CLEAR ENTRY BUFFER command.

SYNTAX

Return String=ENTRY$()

GET CLIPBOARD$
This command will read a string stored in the system clipboard. The system clipboard remains in tact even after the program has been terminated and can be used to transfer values from program to program, or otherwise retain string data beyond the scope of the program.

SYNTAX

Return String=GET CLIPBOARD$()

GET REGISTRY
Get a value from the specified registry location. The folder name points to the general area within the registry. The key name points to the label that describes the data stored within the registry folder.

SYNTAX

Return Integer=GET REGISTRY(Folder Name, Key Name)

GET REGISTRY$
Get a string from the specified registry location. The folder name points to the general area within the registry. The key name points to the label that describes the data stored within the registry folder.

SYNTAX

Return String=GET REGISTRY$(Folder Name, Key Name)

JOYSTICK UP
This command will return an integer value of one if the default joystick is pushed up, otherwise zero is return.

SYNTAX

Return Integer=JOYSTICK UP()

JOYSTICK DOWN
This command will return an integer value of one, if the default joystick is pushed down, otherwise zero is return.

SYNTAX

Return Integer=JOYSTICK DOWN()

JOYSTICK LEFT
This command will return an integer value of one if the default joystick is pushed left, otherwise zero is return.

SYNTAX

Return Integer=JOYSTICK LEFT()

JOYSTICK RIGHT
This command will return an integer value of one if the default joystick is pushed right, otherwise zero is return.

SYNTAX

Return Integer=JOYSTICK RIGHT()

JOYSTICK X
This command will return the X axis value of the default analogue joystick.

SYNTAX
Return Integer=JOYSTICK X()

JOYSTICK Y
This command will return the Y axis value of the default analogue joystick.

SYNTAX
Return Integer=JOYSTICK Y()

JOYSTICK Z
This command will return the Z axis value of the default analogue joystick.

SYNTAX
Return Integer=JOYSTICK Z()

JOYSTICK FIRE A
This command will return an integer value of one if the default joystick fire button A is pressed, otherwise zero will be returned.

SYNTAX
Return Integer=JOYSTICK FIRE A()

JOYSTICK FIRE B
This command will return an integer value of one if the default joystick fire button B is pressed, otherwise zero will be returned.

SYNTAX
Return Integer=JOYSTICK FIRE B()

JOYSTICK FIRE C
This command will return an integer value of one if the default joystick fire button C is pressed, otherwise zero will be returned.

SYNTAX
Return Integer=JOYSTICK FIRE C()

JOYSTICK FIRE D
This command will return an integer value of one if the default joystick fire button D is pressed, otherwise zero will be returned.

SYNTAX
Return Integer=JOYSTICK FIRE D()

JOYSTICK FIRE X
This command will give support to read up to 32 fire buttons.

SYNTAX
Return Integer=JOYSTICK FIRE X(Button Number)

JOYSTICK SLIDER A
This command will return the Slider A value of the default analogue joystick.

SYNTAX

Return Integer=JOYSTICK SLIDER A()

JOYSTICK SLIDER B
This command will return the Slider B value of the default analogue joystick.

SYNTAX

Return Integer=JOYSTICK SLIDER B()

JOYSTICK SLIDER C
This command will give support for a third slider value.

SYNTAX

Return Integer=JOYSTICK SLIDER C()

JOYSTICK SLIDER D
This command will give support for a forth slider value.

SYNTAX

Return Integer=JOYSTICK SLIDER D()

JOYSTICK TWIST X
This command will give support for reading joystick X twist.

SYNTAX

Return Integer=JOYSTICK TWIST X()

JOYSTICK TWIST Y
This command will give support for reading joystick Y twist.

SYNTAX

Return Integer=JOYSTICK TWIST Y()

JOYSTICK TWIST Z
This command will give support for reading joystick Z twist.

SYNTAX

Return Integer=JOYSTICK TWIST Z()

JOYSTICK HAT ANGLE
This command will give support for reading up to 4 HAT controllers. When the hat controller is neutral, a value of -1 is returned. When the hat is being used, the return value is in tens of a degree so north would be zero, east would be 900, south is 1800 and west is 2700.

SYNTAX

Return Integer=JOYSTICK HAT ANGLE(*Hat Number*)

CONTROL DEVICE NAME$
This command will return a string of the name of the current control device.

SYNTAX

Return String=CONTROL DEVICE NAME$()

CONTROL DEVICE X
This command will return the X axis value of the current control device.

SYNTAX

```
Return Integer=CONTROL DEVICE X()
```

CONTROL DEVICE Y
This command will return the Y axis value of the current control device.

SYNTAX

```
Return Integer=CONTROL DEVICE Y()
```

CONTROL DEVICE Z
This command will return the Z axis value of the current control device.

SYNTAX

```
Return Integer=CONTROL DEVICE Z()
```

FILE COMMAND SET

These commands provide functionality for the loading, modification and saving of files. Create, search, delete, move, rename and copy both files and directories with a few simple commands.

DIR
This command will output the entire contents of the current working directory to the screen. The command serves little use for effective file scanning, but provides a simple way to view files.

SYNTAX
```
DIR
```

DRIVELIST
This command will output the available drives to the screen. The command serves little use for effective drive scanning, but provides a simple way to view available drives.

SYNTAX
```
DRIVELIST
```

SET DIR
This command will set the current working directory to the specified path. The path can be absolute or relative. Absolute paths contain the entire path including the drive letter. A relative path assumes the program has a valid current working directory and continues the path from the current location.

SYNTAX
```
SET DIR Path$
```

MAKE FILE
This command will create an empty file. The filename must not exist or the command will fail.

SYNTAX
```
MAKE FILE Filename
```

MAKE DIRECTORY
This command will create an empty directory. The directory name must not exist or the command will fail.

SYNTAX
```
MAKE DIRECTORY Directory Name
```

DELETE FILE
This command will delete an existing file. The file must exist or the command will fail.

SYNTAX
```
DELETE FILE Filename
```

DELETE DIRECTORY
This command will delete an existing directory. The directory must exist or the command will fail. The directory must be completely empty or this command will not succeed.

SYNTAX
```
DELETE DIRECTORY Directory Name
```

COPY FILE
This command will copy an existing file to a new file. The destination filename must not exist or the command will fail.

SYNTAX

```
COPY FILE Source Filename, Destination Filename
```

MOVE FILE
This command will move an existing file to a new location. The destination filename must not exist or the command will fail.

SYNTAX
```
MOVE FILE Source Filename, Destination Filename
```

RENAME FILE
This command will rename an existing file to a new name. The new filename must not exist or the command will fail.

SYNTAX
```
RENAME FILE Source Filename, New Filename
```

EXECUTE FILE
This command will shell execute a file. The commandline is used to pass additional data into the file being executed. The directory is used to optionally specify a directory other than the current directory. The file must exist or the command will fail. Passing a document, rather than an executable as the filename will cause the document to be opened. If the optional Wait Flag value is set to one, then the application will wait until the executable has finished.

SYNTAX
```
EXECUTE FILE Filename, Commandline, Directory
EXECUTE FILE Filename, Commandline, Directory, Wait Flag
```

FIND FIRST
This command will begin a file search by locating the first file in the current working directory. If this command succeeds, a file will be stored internally and its data can be extracted using the GET FILE NAME$(), GET FILE DATE$() and GET FILE TYPE() commands.

SYNTAX
```
FIND FIRST
```

FIND NEXT
This command will continue a file search by locating the next file in the current working directory. If this command succeeds, a file will be stored internally and its data can be extracted using the GET FILE NAME$(), GET FILE DATE$() and GET FILE TYPE() commands. A file search can be started with the FIND FIRST command.

SYNTAX
```
FIND NEXT
```

CD
This command will set the current working directory to the specified path. The path can be absolute or relative. Absolute paths contain the entire path including the drive letter. A relative path assumes the program has a valid current working directory and continues the path from the current location.

SYNTAX
```
CD Path$
```

OPEN TO READ
This command will open a file, ready for reading. The file must exist or the command will fail. You can open up to 32 files at the same time, using a file number range of 1 through to 32.

SYNTAX
```
OPEN TO READ File Number, Filename
```

OPEN TO WRITE
This command will open a file, ready for writing. The file must not exist or the command will fail. You can open up to 32 files

at the same time, using a file number range of 1 through to 32.

SYNTAX

OPEN TO WRITE File Number, Filename

CLOSE FILE
This command will close a file that has been previously opened using OPEN TO READ or OPEN TO WRITE. The file must be open or the command will fail.

SYNTAX

CLOSE FILE File Number

READ BYTE
This command will read a byte of data from the file and store it as an integer value in the variable specified. The file specified by the file number must be open or the command will fail.

SYNTAX

READ BYTE File Number, Variable

READ FILE
This command will read a long word of data from the file and store it as an integer value in the variable specified. This is the standard command for reading data where the datatype is not important. The file specified by the file number must be open or the command will fail.

SYNTAX

READ FILE File Number, Variable

READ DIRBLOCK
This command extracts an entire directory from a pack file. A pack file is like a normal file you create yourself using the OPEN and CLOSE commands, but has the additional feature that as well as storing numerics and strings, you can also store entire files and directories.

SYNTAX

READ DIRBLOCK File Number, Folder to Read

READ FILEBLOCK
This command will extract an entire file from a pack file. A pack file is like a normal file you create yourself using the OPEN and CLOSE commands, but have the additional feature that as well as storing numerics and strings, you can also store entire files and directories.

SYNTAX

READ FILEBLOCK File Number, Filename to Read

READ FLOAT
This command will read a float from the file and store it as a real value in the real variable specified. The file specified by the file number must be open or the command will fail.

SYNTAX

READ FLOAT File Number, Variable

READ LONG
This command will read a long word of data from the file and store it as an integer value in the variable specified. A long word represents four bytes. The file specified by the file number must be open or the command will fail.

SYNTAX

READ LONG File Number, Variable

READ MEMBLOCK

Create a memblock from the currently open file. The file must contain a memblock created with the WRITE MEMBLOCK command, at the exact position within the file. You must specify the file and memblock numbers using integer values.

SYNTAX

READ MEMBLOCK File Number, Memblock Number

READ STRING
This command will read a string from the file and store it as a string in the variable specified. The file specified by the file number must be open or the command will fail.

SYNTAX

READ STRING File Number, Variable String

READ WORD
This command will read a word of data from the file and store it as an integer value in the variable specified. A word represents two bytes. The file specified by the file number must be open or the command will fail.

SYNTAX

READ WORD File Number, Variable

WRITE WORD
This command will write a word of data to the file from an integer value. A word represents two bytes. The file specified by the file number must be open or the command will fail.

SYNTAX

WRITE WORD File Number, Variable

WRITE BYTE
This command will write a byte of data to the file from an integer value. The file specified by the file number must be open or the command will fail.

SYNTAX

WRITE BYTE File Number, Variable

WRITE LONG
This command will write a long word of data to the file from an integer value. A long word represents four bytes. The file specified by the file number must be open or the command will fail.

SYNTAX

WRITE LONG File Number, Variable

WRITE DIRBLOCK
This command will write an entire directory to a pack file. A pack file is like a normal file you create yourself using the OPEN and CLOSE commands, but has the additional feature that as well as storing numerics and strings, you can also store entire files and directories.

SYNTAX

WRITE DIRBLOCK File Number, Folder to Create

WRITE FILEBLOCK
This command will write a file to a pack file. A pack file is like a normal file you create yourself using the OPEN and CLOSE commands, but has the additional feature that as well as storing numerics and strings, you can also store entire files and directories.

SYNTAX

WRITE FILEBLOCK File Number, Filename to Create

WRITE FLOAT

This command will write a float to the file from a real value. The file specified by the file number must be open or the command will fail.

SYNTAX

```
WRITE FLOAT File Number, Variable
```

WRITE FILE
This command will write a long word of data to the file from an integer value. This is the standard command for writing data where the datatype is not important. The file specified by the file number must be open or the command will fail.

SYNTAX

```
WRITE FILE File Number, Variable
```

WRITE MEMBLOCK
Write the specified memblock to a file open for writing. You can store multiple memblocks within a currently open file, and is useful for creating your own file formats. To retrieve the memblock you must use the READ MEMBLOCK command. You must specify the file and memblock numbers using integer values.

SYNTAX

```
WRITE MEMBLOCK File Number, Memblock Number
```

WRITE STRING
This command will write the specified string to the file. The string will be terminated in the file with the standard carriage return ASCII characters (13)+(10). The file specified by the file number must be open or the command will fail.

SYNTAX

```
WRITE STRING File Number, String
```

SKIP BYTES
This command will skip the specified number of bytes in a current open file, opened using the OPEN TO READ command. You would use this command where you have fore-knowledge of the file contents.

SYNTAX

```
SKIP BYTES File Number, Bytes To Skip
```

WRITE BYTE TO FILE
This command will write a single byte to the specified file. The position is specified in bytes from the beginning of the filedata. You can use this to modify a file at the byte level.

SYNTAX

```
WRITE BYTE TO FILE Filename, Position, ByteValue
```

READ BYTE FROM FILE
This command will read a single byte from the specified file. The position is specified in bytes from the beginning of the filedata. You can use this to obtain byte perfect reading of any file.

SYNTAX

```
Return Integer=READ BYTE FROM FILE(Filename, Position)
```

PERFORM CHECKLIST FOR DRIVES
This command will output the available drives to a checklist. The command provides a convenient way to store the available drives without using additional data structures, but does tie up the checklist when in use. Use the CHECKLIST commands in the SYSTEM command set to access the checklist.

SYNTAX

```
PERFORM CHECKLIST FOR DRIVES
```

PERFORM CHECKLIST FOR FILES

This command will output the contents of the current working directory to a checklist. The command provides a convenient way to store the contents of a directory without using additional data structures, but does tie up the checklist when in use. Use the CHECKLIST commands in the SYSTEM command set to access the checklist.

SYNTAX

```
PERFORM CHECKLIST FOR FILES
```

MAKE MEMBLOCK FROM FILE
Create a memblock from the file specified. The memblock will be the same size as the currently open file, and contain the entire data of the file specified. You must specify the file and memblock numbers using integer values.

SYNTAX

```
MAKE MEMBLOCK FROM FILE Memblock Number, File Number
```

MAKE FILE FROM MEMBLOCK
Create a file from a memblock, allowing you to directly create an arrangement of bytes in a memblock and create an exact file layout with no additional bytes written.

SYNTAX

```
MAKE FILE FROM MEMBLOCK File Number, Memblock Number
```

FILE EXIST
This command will return an integer value of one if the specified file exists, otherwise zero is returned.

SYNTAX

```
Return Integer=FILE EXIST(Filename)
```

PATH EXIST
This command will return an integer value of one if the specified path exists, otherwise zero is returned. You can use this command to check whether a directory exists.

SYNTAX

```
Return Integer=PATH EXIST(Path String)
```

GET DIR$
This command will get the absolute path of the current working directory. Absolute paths contain the entire path including the drive letter.

SYNTAX

```
Return String=GET DIR$()
```

APPNAME$
This command will return the executable name of the running application.

SYNTAX

```
Return String=APPNAME$()
```

WINDIR$
This command will return the name of the current windows directory.

SYNTAX

```
Return String=WINDIR$()
```

FILE END
This command will return an integer value of one if the file specified by the file number has no more data to read, otherwise zero is returned.

SYNTAX

Return Integer=FILE END(File Number)

FILE OPEN
This command will return an integer value of one if the file specified by the file number is open, otherwise zero is returned.

SYNTAX
Return Integer=FILE OPEN(File Number)

FILE SIZE
This command will return the size of the specified file in bytes, otherwise zero is returned. The file must exist or the command will fail.

SYNTAX
Return Integer=FILE SIZE(Filename)

GET FILE DATE$
This command will get the date string of the file currently recorded by the file search commands FIND FIRST and FIND NEXT.

SYNTAX
Return String=GET FILE DATE$()

GET FILE NAME$
This command will get the filename of the file currently recorded by the file search commands FIND FIRST and FIND NEXT.

SYNTAX
Return String=GET FILE NAME$()

GET FILE TYPE
This command will get the type value of the file currently recorded by the file search commands FIND FIRST and FIND NEXT. A type value of zero indicates it is a file. A type value of 1 indicates it is a directory. A type value of -1 indicates there are no more files in the current working directory.

SYNTAX
Return Integer=GET FILE TYPE()

GET FILE CREATION$
This command will return the creation date of a file currently pointed to by the use of the FILE FIRST and FILE NEXT commands.

SYNTAX
Return String=GET FILE CREATION$()

DISPLAY COMMAND SET

These commands provide functionality for the control of the application display. With them you can control the screen resolution, window settings and display adapter selection.

PERFORM CHECKLIST FOR GRAPHICS CARDS
This command will make a checklist of all installed display cards. Some system have more than one 3D accelerator installed and often provide different feature sets and levels of performance. Use the CHECKLIST commands in the SYSTEM command set to access the checklist.

SYNTAX
```
PERFORM CHECKLIST FOR GRAPHICS CARDS
```

PERFORM CHECKLIST FOR DISPLAY MODES
This command will scan your system and make a checklist of all display modes your video card can handle. Use the CHECKLIST commands in the SYSTEM command set to access the checklist.

SYNTAX
```
PERFORM CHECKLIST FOR DISPLAY MODES
```

SET DISPLAY MODE
This command will set the screen display mode if it is available on the current graphics card. If this command fails, it will only generate a runtime warning in the CLI. A final EXE produced with this command will not cause your program to fail on machines that do not support the resolution. It is recommended, however, that you check the availability of the display mode before using this command. You can use the CHECK DISPLAY MODE() command to see whether the display mode is supported.

SYNTAX
```
SET DISPLAY MODE Width, Height, Depth
```

SET GRAPHICS CARD
This command will set the current display card as described by the name provided. The display card name can be found by performing a PERFORM CHECKLIST FOR GRAPHICS CARDS.

SYNTAX
```
SET GRAPHICS CARD Cardname
```

SET GAMMA
This command will set the screens red, green and blue gamma levels. You can change the gamma to fade in and out the contents of the screen or alter the ratio of colours displayed. The red, green and blue component values can range from 0 to 511, with 255 being the default values. Reducing these values fades each colour component out of the screen, and above the default value enhances the ratio of the component colour. Some graphics cards do not support gamma alteration.

SYNTAX
```
SET GAMMA Red, Green, Blue
```

SHOW WINDOW
This command will show the window.

SYNTAX
```
SHOW WINDOW
```

HIDE WINDOW
This command will hide the window but keep the application active.

SYNTAX
```
HIDE WINDOW
```

MAXIMIZE WINDOW
This command will maximise the window to the entire desktop screen.

SYNTAX
```
MAXIMIZE WINDOW
```

MINIMIZE WINDOW
This command will minimise the window to a desktop iconized bar.

SYNTAX
```
MINIMIZE WINDOW
```

RESTORE WINDOW
This command will restore the window to the original size and position.

SYNTAX
```
RESTORE WINDOW
```

SET WINDOW ON
This command will activate Windows Mode.

SYNTAX
```
SET WINDOW ON
```

SET WINDOW OFF
This command will deactivate Windows Mode.

SYNTAX
```
SET WINDOW OFF
```

SET WINDOW SIZE
This command will set the windows size.

SYNTAX
```
SET WINDOW SIZE Width, Height
```

SET WINDOW TITLE
This command will change the title of the window.

SYNTAX
```
SET WINDOW TITLE Caption String
```

SET WINDOW LAYOUT
This command will set basic window properties. If the Style Flag is set to one the window is created with titlebar, minimise and close icons and an icon. If the Caption Flag is set to zero, the titlebar is removed from the window. If the Icon Number is zero, the icon reverts to the standard Windows application icon.

SYNTAX
```
SET WINDOW LAYOUT Style, Caption, Icon Number
```

SET WINDOW POSITION
This command will set the window's position.

SYNTAX
```
SET WINDOW POSITION X, Y
```

LOCK BACKBUFFER
This command locks the backbuffer and stores the backbuffer details for direct access. The backbuffer is the actual screen you see each refresh and this command allows you to pass the pointer and other essential data to a DLL for screen modification. Locking the backbuffer prevents other activities to be performed to the backbuffer, so it is essential you unlock the backbuffer when you have finished.

SYNTAX

```
LOCK BACKBUFFER
```

UNLOCK BACKBUFFER
This command unlocks the backbuffer and frees the system to continue as normal. The backbuffer is the actual screen you see each refresh. Locking the backbuffer prevents other activities to be performed regarding the backbuffer so it is essential you unlock the backbuffer when you have finished.

SYNTAX

```
UNLOCK BACKBUFFER
```

SCREEN TYPE
This command will return an integer value of the current screen type. A returned value of 0 indicates the screen is not hardware accelerated. A returned value of 1 indicates the screen is hardware accelerated.

SYNTAX

```
Return Integer=SCREEN TYPE()
```

SCREEN WIDTH
This command will return an integer value of the current screen width.

SYNTAX

```
Return Integer=SCREEN WIDTH()
```

SCREEN HEIGHT
This command will return an integer value of the current screen height.

SYNTAX

```
Return Integer=SCREEN HEIGHT()
```

SCREEN DEPTH
This command will return an integer value of the current screen depth. The depth value indicates the number of bits used to make up a color for the screen and therefore reveal how many colors in total can be used by the screen. A value of 16 indicates it is a 16-bit screen and uses 32000 colors, whereas a 32-bit screen uses 16 million colors.

SYNTAX

```
Return Integer=SCREEN DEPTH()
```

SCREEN INVALID
This command will return a one if the application has been switched out and back in.

SYNTAX

```
Return Integer=SCREEN INVALID()
```

SCREEN FPS
This command will get the current frames per second to measure how many times the screen is refreshed each second. The integer value returned is measured in units of 1/1000th of a second.

SYNTAX

```
Return Integer=SCREEN FPS()
```

CURRENT GRAPHICS CARD$
This command will return the name of the current display card being used by the system.

SYNTAX
Return String=CURRENT GRAPHICS CARD$()

CHECK DISPLAY MODE
This command will check that the display mode you have specified is available on the current system. The parameters should use integer values.

SYNTAX
Return Integer=CHECK DISPLAY MODE(Width, Height, Depth)

GET BACKBUFFER PTR
This command will return the actual pointer to the backbuffer. You can pass the pointer to a DLL which can directly access the memory of the backbuffer. You can only use this command when you have used the LOCK BACKBUFFER command.

SYNTAX
Return Integer=GET BACKBUFFER PTR()

GET BACKBUFFER WIDTH
This command will return the width of the backbuffer. You can pass this data to a DLL to assist in the direct access of backbuffer memory. You can only use this command when you have used the LOCK BACKBUFFER command.

SYNTAX
Return Integer=GET BACKBUFFER WIDTH()

GET BACKBUFFER HEIGHT
This command will return the height of the backbuffer. You can pass this data to a DLL to assist in the direct access of backbuffer memory. You can only use this command when you have used the LOCK BACKBUFFER command.

SYNTAX
Return Integer=GET BACKBUFFER HEIGHT()

GET BACKBUFFER DEPTH
This command will return the depth of the backbuffer. You can pass this data to a DLL to assist in the direct access of backbuffer memory. You can only use this command when you have used the LOCK BACKBUFFER command.

SYNTAX
Return Integer=GET BACKBUFFER DEPTH()

GET BACKBUFFER PITCH
This command will return the pitch of the backbuffer. You can pass this data to a DLL to assist in the direct access of backbuffer memory. The pitch is similar to the width of the backbuffer, and may be larger should the backbuffer use a cache at the end of each horizontal line. You can only use this command when you have used the LOCK BACKBUFFER command.

SYNTAX
Return Integer=GET BACKBUFFER PITCH()

BASIC2D COMMAND SET

These commands allow you to draw simple 2D shapes to the screen. Although more modern applications rely on importing image files for graphic content, there are some cases where you need to draw to the screen directly.

INK
This command will set the current ink color using the specified color value. color values range from 0 to over 16 million, that represent every combination of red, green and blue to make up the final color. You can use the RGB command to make the generation of the color value straight forward. The parameter should be specified using integer values.

SYNTAX

```
INK Foreground Color,Background Color
```

CLS
This command will clear the screen using the specified color value. Color values range from 0 to over 16 million, that represent every combination of red, green and blue to make up the final Color. You can use the RGB command to make the generation of the Color value straight forward.

SYNTAX

```
CLS Color Value
```

DOT
This command will put a pixel on the screen in the current ink color. The command requires the Coordinates to place the pixel on the screen. The parameters should be specified using integer values. You can also draw to a bitmap, by using the SET CURRENT BITMAP command.

SYNTAX

```
DOT X,Y
DOT X,Y,Color Value
```

LINE
This command will put a line on the screen in the current ink color. The command requires two sets of Coordinates to draw a line from one to the other on the screen. The parameters should be specified using integer values. You can also draw to a bitmap, by using the SET CURRENT BITMAP command.

SYNTAX

```
LINE X1,Y1,X2,Y2
```

BOX
This command will draw a filled box on screen in the current ink color. The command requires the top left and bottom right coordinates of the box. The parameters should be specified using integer values. You can also draw to a bitmap, by using the SET CURRENT BITMAP command.

SYNTAX

```
BOX Left,Top,Right,Bottom
BOX Left, Top, Right, Bottom, Color1, Color2, Color3, Color4
```

CIRCLE
This command will draw a circle on screen using the current ink color. The command requires the radius and the Coordinates that will represent the center of the circle to be drawn. The parameters should be specified using integer values. You can also draw to a bitmap, by using the SET CURRENT BITMAP command.

SYNTAX

```
CIRCLE X,Y,Radius
```

ELLIPSE
This command will draw an ellipse on screen using the current ink color. The command requires the X-Radius, Y-Radius and the Coordinates that will represent the center of the ellipse to be drawn. The parameters should be specified using integer values. You can also draw to a bitmap, by using the SET CURRENT BITMAP command.

SYNTAX

```
ELLIPSE X,Y,X Radius,Y Radius
```

LOCK PIXELS
This command will lock the current bitmap for faster reading and writing of visual data. Rather than use DOT and POINT to manipulate pixel data, use this command in combination with GET PIXELS POINTER to write directly to the memory storing the visual data.

SYNTAX

```
LOCK PIXELS
```

UNLOCK PIXELS
This command will unlock the current bitmap after faster reading and writing of visual data. Rather than use DOT and POINT to manipulate pixel data, use the LOCK PIXELS command to perform modification of visual data much faster.

SYNTAX

```
UNLOCK PIXELS
```

RGB
This command will return the final color value of a combination of red, green and blue intensities. For each of the Red, Green and Blue components you must enter a value between 0 and 255. All zero will return a color value that represents black. All 255 will return a color value that represents white. By setting the Red component to 255 and the rest to zero, the command will return a color value that represents red. The parameters should be specified using integer values.

SYNTAX

```
Return DWORD=RGB(Red Value,Green Value,Blue Value)
```

RGBR
This command will extract the red component value from the specified RGB value. You can generate an RGB value from color components using the RGB command.

SYNTAX

```
Return Integer=RGBR(Red Value)
```

RGBG
This command will extract the green component value from the specified RGB value. You can generate an RGB value from color components using the RGB command.

SYNTAX

```
Return Integer=RGBG(Green Value)
```

RGBB
This command will extract the blue component value from the specified RGB value. You can generate an RGB value from color components using the RGB command.

SYNTAX

```
Return Integer=RGBB(Blue Value)
```

POINT
This command will return the pixel color value from the screen at the specified Coordinates. The parameters should be specified using integer values. You can also read from bitmaps by using the SET CURRENT BITMAP command.

SYNTAX

```
Return DWORD=POINT(X,Y)
```

GET PIXELS POINTER
This command in combination with LOCK PIXELS will return the pointer to the first pixel in the visual surface. You can use the indirect symbol to write and read using the value of this pointer. The indirect symbol is specified by placing a * character before the variable holding the pointer.

SYNTAX

Return DWORD=GET PIXELS POINTER()

GET PIXELS PITCH
This command in combination with LOCK PIXELS will return the pitch in bytes of the visual surface you have locked. A pitch is the number of bytes you must skip to get to the next physical line of pixels in a visual surface.

SYNTAX

Return Integer=GET PIXELS PITCH()

BITMAP COMMAND SET

These commands provide functionality for the loading, display, modification and deleting of bitmap graphics. You can load in a variety of picture file formats and position them anywhere on the screen to make your applications more visually appealing.

LOAD BITMAP

This command loads a picture file to the screen. The picture file must be of the BMP, JPG, TGA, DDS, DIB or PNG format. You can optionally provide a Bitmap Number between 0 and 32. Once you have loaded the picture file successfully, you can use the specified bitmap number to modify and manage the bitmap. The bitmap number should be specified using an integer value.

SYNTAX

```
LOAD BITMAP Filename,Bitmap Number
LOAD BITMAP Filename
```

CREATE BITMAP

This command will create a blank bitmap of a specified size. The size of the bitmap is only limited by the amount of system memory available. When you create a bitmap, it becomes the current bitmap. All drawing operations will be re-directed to the current bitmap and away from the screen. You can use the SET CURRENT BITMAP command to restore drawing operations to the screen. The parameters should be specified using integer values.

SYNTAX

```
CREATE BITMAP Bitmap Number,Width,Height
```

DELETE BITMAP

This command will delete a specified Bitmap. Deleting bitmaps that are no longer used greatly improves system performance. The parameter should be specified using an integer value.

SYNTAX

```
DELETE BITMAP Bitmap Number
```

COPY BITMAP

This command will copy the contents of one bitmap into another bitmap providing the destination bitmap is not smaller than the first. The command requires at least a source and destination bitmap. You can optionally specify a source area to be copied from and a destination area to be copied to within each bitmap. If the size of the two areas differ, the source data will be rescaled to fit in the destination area. The parameters should be specified using integer values.

SYNTAX

```
COPY BITMAP From Bitmap,To Bitmap
COPY BITMAP From Bitmap,Left,Top,Right,Bottom,To Bitmap,Left,Top,Right,Bottom
```

FLIP BITMAP

This command will flip the contents of the specified bitmap vertically. The parameter should be specified using an integer value.

SYNTAX

```
FLIP BITMAP Bitmap Number
```

MIRROR BITMAP

This command will mirror the contents of the specified bitmap horizontally. The parameter should be specified using an integer value.

SYNTAX

```
MIRROR BITMAP Bitmap Number
```

FADE BITMAP
This command will fade the contents of a specified Bitmap. You must specify a fade value that sets the level of fading from zero which fades the bitmap completely to black, up to 100 which does not fade the bitmap at all. Fade operations are slow and the completion time depends on the size of the bitmap. The parameters should be specified using integer values.

SYNTAX
```
FADE BITMAP Bitmap Number,Fade Value
```

BLUR BITMAP
This command will blur the contents of a specified Bitmap. You must specify a blur value from 1 to 6 to provide the intensity of the blurring. A blur value of 1 will perform mild blurring, up to a value of 6 that causes severe blurring. The greater the intensity of blurring, the longer it takes to perform. The time it takes to blur a bitmap is also dependent on the size of the bitmap. The parameters should be specified using integer values.

SYNTAX
```
BLUR BITMAP Bitmap Number,Blur Value
```

SET CURRENT BITMAP
This command will set the current bitmap number for all drawing operations. Use this command if you wish to draw, paste and extract images from the bitmap. Setting the current bitmap to zero points all drawing operations to the screen. The parameter should be specified using an integer value.

SYNTAX
```
SET CURRENT BITMAP Bitmap Number
```

CURRENT BITMAP
This command will return an integer value of the current bitmap number being used. If this value is zero, the screen is the current bitmap and drawing operations are performed on the visible screen. The parameter should be specified using an integer value.

SYNTAX
```
Return Integer=CURRENT BITMAP()
```

BITMAP EXIST
This command will return a one if the specified bitmap exists, otherwise zero is returned. The parameter should be specified using an integer value.

SYNTAX
```
Return Integer=BITMAP EXIST(Bitmap Number)
Return Integer=BITMAP EXIST()
```

BITMAP WIDTH
This command will return an integer value of the width of the current bitmap. You can optionally provide a bitmap number to return the width of a specified bitmap. The parameter should be specified using an integer value.

SYNTAX
```
Return Integer=BITMAP WIDTH(Bitmap Number)
Return Integer=BITMAP WIDTH()
```

BITMAP HEIGHT
This command will return an integer value of the height of the current bitmap. You can optionally provide a bitmap number to return the height of a specified bitmap. The parameter should be specified using an integer value.

SYNTAX
```
Return Integer=BITMAP HEIGHT(Bitmap Number)
Return Integer=BITMAP HEIGHT()
```

BITMAP DEPTH

This command will return an integer value of the color bit-depth of the current bitmap. You can optionally provide a bitmap number to return the color bit-depth of a specified bitmap. color bit-depths represent the maximum amount of colors the bitmap can hold. The parameter should be specified using an integer value.

SYNTAX

```
Return Integer=BITMAP DEPTH(Bitmap Number)
Return Integer=BITMAP DEPTH()
```

BITMAP MIRRORED

This command will return a one if the current bitmap has been mirrored, otherwise zero is returned. You can optionally provide a bitmap number to check whether a specified bitmap has been mirrored. The parameter should be specified using an integer value.

SYNTAX

```
Return Integer=BITMAP MIRRORED(Bitmap Number)
Return Integer=BITMAP MIRRORED()
```

BITMAP FLIPPED

This command will return a one if the current bitmap has been flipped, otherwise zero is returned. You can optionally provide a bitmap number to check whether a specified bitmap has been flipped. The parameter should be specified using an integer value.

SYNTAX

```
Return Integer=BITMAP FLIPPED(Bitmap Number)
Return Integer=BITMAP FLIPPED()
```

SOUND COMMAND SET

These commands provide functionality to load, control and delete sound files. Sounds are loaded in the WAV file format. A common set of controls exist to play, stop, pause, resume, slow, speedup and reduce in volume your sound effects. In addition, you can play the sound in 3D to create atmosphere within your games.

LOAD SOUND
This command will load a WAV sound file into the specified Sound Number. The Sound Number must be an integer value. To load other sound formats, use LOAD MUSIC or in extreme cases LOAD ANIMATION.

SYNTAX

```
LOAD SOUND Filename,Sound Number
LOAD SOUND Filename, Sound Number, Flag
```

LOAD 3DSOUND
This command will load a WAV sound file into the specified Sound Number as a special 3D sound. The specified WAV sound file must by Mono or the load will fail. 3D sounds can be placed in 3D space and heard through the virtual ears of a listener. The listener can also be placed anywhere in 3D space creating true surround sound capabilities. The Sound Number must be an integer value.

SYNTAX

```
LOAD 3DSOUND Filename,Sound Number
```

SAVE SOUND
This command will save a sound to a file previously captured using the RECORD SOUND command. You cannot save a sound file that originated from any other source.

SYNTAX

```
SAVE SOUND Filename,Sound Number
```

DELETE SOUND
This command will delete the specified sound previously loaded into Sound Number.

SYNTAX

```
DELETE SOUND Sound Number
```

CLONE SOUND
This command will clone a sound into the specified Destination Sound Number. Cloning a sound will create a new sound that can be played like any other loaded sound, but uses the same WAV data of the original sound. The advantage of sound cloning is that one hundred sounds could be used with only a single instance of the sound data stored in memory. The Sound Number must be an integer value.

SYNTAX

```
CLONE SOUND Destination Sound,Source Sound
```

PLAY SOUND
This command will play the specified Sound Number. An optional parameter allows you to specify a start position in bytes that skips the initial part of the sample to be played.

SYNTAX

```
PLAY SOUND Sound Number
PLAY SOUND Sound Number,Start Position
```

LOOP SOUND
This command will play and loop the specified Sound Number continuously. Optional parameters allow you to specify a start

position, end position and initial position in bytes that a looping sound will use as it plays.

SYNTAX

```
LOOP SOUND Sound Number
LOOP SOUND Sound Number,Start Position
LOOP SOUND Sound Number,Start Position,End Position
LOOP SOUND Sound Number,Start Position,End Position,Initial Position
```

STOP SOUND
This command will stop the specified Sound Number if it is playing.

SYNTAX

```
STOP SOUND Sound Number
```

PAUSE SOUND
This command will pause the specified Sound Number whilst it is playing.

SYNTAX

```
PAUSE SOUND Sound Number
```

RESUME SOUND
This command will resume the specified Sound Number after it has been paused.

SYNTAX

```
RESUME SOUND Sound Number
```

POSITION SOUND
This command will position the specified 3D sound in 3D space. The 3D sounds you hear are calculated based on the position of the sound and the listener.

SYNTAX

```
POSITION SOUND Sound Number,X,Y,Z
```

RECORD SOUND
This command will start recording a sound from the microphone. You can optionally specify an additional integer parameter to record a sound for any amount of seconds. The default if this parameter is not specified is five seconds of recording. You must specify an empty sound number using an integer value.

SYNTAX

```
RECORD SOUND Sound Number
RECORD SOUND Sound Number, Duration
```

STOP RECORDING SOUND
This command will stop recording a sound previously started using RECORD SOUND.

SYNTAX

```
STOP RECORDING SOUND
```

SET SOUND PAN
This command will set the pan of standard sounds by locating it between the left and right speakers. A negative value will move the sound to the left speaker, a positive value will move it to the right. Sound panning does not work with 3D sounds. The pan value must be an integer value between -10,000 and 10,000.

SYNTAX

```
SET SOUND PAN Sound Number,Pan Value
```

SET SOUND SPEED
This command will set the frequency used by the specified Sound Number. Decibel frequency ranges from 100 to 100,000 and must be specified using an integer value.

SYNTAX
```
SET SOUND SPEED Sound Number,Frequency Value
```

SET SOUND VOLUME
This command will set the percentage volume used by the specified Sound Number. The volume value should use an integer value.

SYNTAX
```
SET SOUND VOLUME Sound Number,Volume Number
```

POSITION LISTENER
This command will position the listener in 3D space. The 3D sounds you hear are calculated based on the position of the sound and the listener.

SYNTAX
```
POSITION LISTENER X,Y,Z
```

ROTATE LISTENER
This command will set the direction of the listener. The 3D sounds being played would sound different based on which direction the listener was facing.

SYNTAX
```
ROTATE LISTENER X,Y,Z
```

SCALE LISTENER
This command will scale the listener in 3D space. The 3D sounds you hear are calculated based on the scale as well as the position of the sound and the listener. A value of 1 is the default setting, and a value of 0.5 will make the listener half as sensitive.

SYNTAX
```
SCALE LISTENER Scaling Factor
```

SOUND EXIST
This command will return an integer value of one if the specified Sound Number exists, otherwise zero will be returned.

SYNTAX
```
Return Integer=SOUND EXIST(Sound Number)
```

SOUND TYPE
This command will return an integer value of one if the specified Sound Number is a special 3D sound, otherwise zero will be returned.

SYNTAX
```
Return Integer=SOUND TYPE(Sound Number)
```

SOUND PLAYING
This command will return an integer value of one if the specified Sound Number is playing, otherwise zero will be returned.

SYNTAX
```
Return Integer=SOUND PLAYING(Sound Number)
```

SOUND LOOPING
This command will return an integer value of one if the specified Sound Number is looping, otherwise zero will be returned.

SYNTAX

Return Integer=SOUND LOOPING(Sound Number)

SOUND PAUSED
This command will return an integer value of one if the specified Sound Number is paused, otherwise zero will be returned.

SYNTAX

Return Integer=SOUND PAUSED(Sound Number)

SOUND POSITION X
This command will return the current X position of the 3D sound specified by Sound Number.

SYNTAX

Return Float=SOUND POSITION X(Sound Number)

SOUND POSITION Y
This command will return the current Y position of the 3D sound specified by Sound Number.

SYNTAX

Return Float=SOUND POSITION Y(Sound Number)

SOUND POSITION Z
This command will return the current Z position of the 3D sound specified by Sound Number.

SYNTAX

Return Float=SOUND POSITION Z(Sound Number)

SOUND PAN
This command will return the current pan value of the specified Sound Number.

SYNTAX

Return Integer=SOUND PAN(Sound Number)

SOUND SPEED
This command will return the current frequency of the specified Sound Number.

SYNTAX

Return Integer=SOUND SPEED(Sound Number)

SOUND VOLUME
This command will return the current percentage volume of the specified Sound Number.

SYNTAX

Return Integer=SOUND VOLUME(Sound Number)

LISTENER POSITION X
This command will return the current X position of the listener.

SYNTAX

Return Float=LISTENER POSITION X()

LISTENER POSITION Y
This command will return the current Y position of the listener.

SYNTAX

Return Float=LISTENER POSITION Y()

LISTENER POSITION Z
This command will return the current Z position of the listener.

SYNTAX
Return Float=LISTENER POSITION Z()

LISTENER ANGLE X
This command will return the current X angle of the listeners direction.

SYNTAX
Return Float=LISTENER ANGLE X()

LISTENER ANGLE Y
This command will return the current Y angle of the listeners direction.

SYNTAX
Return Float=LISTENER ANGLE Y()

LISTENER ANGLE Z
This command will return the current Z angle of the listeners direction.

SYNTAX
Return Float=LISTENER ANGLE Z()

MUSIC COMMAND SET

These commands provide functionality to load, control and delete music files. You can load in a variety of music file formats, and they all share a common set of controls to play, stop, pause, resume, slow, speedup and reduce in volume any kind of music. You can also play a music track from any Music CD present on your system.

LOAD MUSIC
This command will load a music file into the specified music number. The music file must be of the MIDI or MP3 format. The music number should be an integer value.

SYNTAX
```
LOAD MUSIC Filename,Music Number
```

LOAD CDMUSIC
This command will play CD Audio music stored on your CD. The CD Audio track must be specified and loaded before it can be played. Only one CD Audio track can be loaded and played at any one time. To play a new track, you must delete a previously loaded track before loading the new one. The parameters must be specified using integer values.

SYNTAX
```
LOAD CDMUSIC Track Number,Music Number
```

DELETE MUSIC
This command will delete the specified music previously loaded into a music number.

SYNTAX
```
DELETE MUSIC Music Number
```

PLAY MUSIC
This command will play the specified music number.

SYNTAX
```
PLAY MUSIC Music Number
```

LOOP MUSIC
This command will play and loop the specified music continuously.

SYNTAX
```
LOOP MUSIC Music Number
```

STOP MUSIC
This command will stop the specified music number if it is playing.

SYNTAX
```
STOP MUSIC Music Number
```

PAUSE MUSIC
This command will pause the specified music number if it is playing.

SYNTAX
```
PAUSE MUSIC Music Number
```

RESUME MUSIC
This command will resume the specified music number if it currently paused.

SYNTAX

```
RESUME MUSIC Music Number
```

SET MUSIC SPEED
This command will set the speed at which the music is playing. The default value is 100, as a percentage of the standard rate of play. A value of 50 means the music is playing at half the normal speed where 200 will play the music twice as fast.

SYNTAX
```
SET MUSIC SPEED Music Number,Speed
```

SET MUSIC VOLUME
This command will set the volume at which the music is playing. The default value is 100, as a percentage of the standard volume of play. A value of 50 means the music is playing at half the normal volume where 200 will play the music twice as loud.

SYNTAX
```
SET MUSIC VOLUME Music Number,Volume
```

MUSIC EXIST
This command will return an integer value of one if the specified music exists, otherwise zero is returned.

SYNTAX
```
Return Integer=MUSIC EXIST(Music Number)
```

MUSIC PLAYING
This command will return an integer value of one if the specified music is playing, otherwise zero is returned.

SYNTAX
```
Return Integer=MUSIC PLAYING(Music Number)
```

MUSIC LOOPING
This command will return an integer value of one if the specified music is looping, otherwise zero is returned.

SYNTAX
```
Return Integer=MUSIC LOOPING(Music Number)
```

MUSIC PAUSED
This command will return an integer value of one if the specified music is paused, otherwise a zero is returned.

SYNTAX
```
Return Integer=MUSIC PAUSED(Music Number)
```

MUSIC SPEED
This command will return an integer value representing the speed at which the music is playing. The default value is 100, as a percentage of the standard rate. A value of 50 means the music is playing at half the normal speed.

SYNTAX
```
Return Integer=MUSIC SPEED(Music Number)
```

MUSIC VOLUME
This command will return an integer value representing the volume at which the music is playing. The default value is 100, as a percentage of the standard volume. A value of 50 means the music is playing at half the normal volume.

SYNTAX
```
Return Integer=MUSIC VOLUME(Music Number)
```

GET NUMBER OF CD TRACKS

This command will return the number of CD tracks currently available from the CD Audio media present on the system. If no CD Audio media is present, this command returns a zero.

SYNTAX

Return Integer=GET NUMBER OF CD TRACKS()

SPRITE COMMAND SET

These commands provide functionality to create, control and delete sprites. Sprites are flat images that are drawn onto the screen and overlap other graphics such as 2D art and text. You can have many thousands of sprites on the screen based on the speed of your system and available hardware. Sprites can be controlled to rotate, scale, fade and change colour to add visual effect to your games.

SPRITE
This command will set the position and image number of the specified sprite. Providing you are using a valid image number from a previous call to the GET IMAGE command, and the position of the sprite is in the screen area, you will see your sprite displayed. You can move your sprite by calling the sprite command with new position coordinates. You can animate your sprite by calling this command with different image numbers to create the effect of animation. You are able to have over sixty thousand sprites on the screen at any one time, but it is advisable to restrict yourself to a few hundred sprites for speed critical programs. The parameters should be specified using integer values.

SYNTAX
```
SPRITE Sprite Number, XPos, YPos, Image Number
```

SET SPRITE
This command will set whether the specified sprite restores its background and whether background transparency is ignored. If the backsave state is set to zero, the sprite will not restore its background and leave a trail as it moves. If the transparency state is set to zero, the sprite will not treat black as a transparent color. A transparent color in the sprite image does not write to the screen. If this feature is disabled, the sprite would appear as though drawn inside a black rectangle where transparency had previously been used. Both states are set to one as these are the most common settings. If you set the backsave state to zero, it is your responsibility to clear or paste the background each time the sprite is moved or animated. The sprite number should be specified using an integer value. The backsave and transparency states should be specified as either zero or one.

SYNTAX
```
SET SPRITE Sprite Number, Backsave, Transparency
```

DELETE SPRITE
This command will delete the specified sprite from memory. Deleting unused sprites increases system performance. The parameter should be specified using an integer value.

SYNTAX
```
DELETE SPRITE Sprite Number
```

CLONE SPRITE
This command will clone the specified sprite to the target sprite number. A cloned sprite shares all the settings of the original sprite, and once created can be modified as a sprite in its own right. The parameters should be specified using integer values.

SYNTAX
```
CLONE SPRITE Sprite Number, Destination Sprite Number
```

SHOW SPRITE
This command will show the specified sprite. The parameter should be specified using an integer value.

SYNTAX
```
SHOW SPRITE Sprite Number
```

HIDE SPRITE
This command will hide the specified sprite. The parameter should be specified using an integer value.

SYNTAX
```
HIDE SPRITE Sprite Number
```

HIDE ALL SPRITES
This command will hide all the sprites currently visible.

SYNTAX
```
HIDE ALL SPRITES
```

SHOW ALL SPRITES
This command will show all the sprites currently invisible.

SYNTAX
```
SHOW ALL SPRITES
```

MOVE SPRITE
This command will move the sprite a specified distance. Used in combination with the ROTATE SPRITE command a very simple top-down player character can be achieved very easily. The velocity must be a real value.

SYNTAX
```
MOVE SPRITE Sprite Number, Velocity
```

OFFSET SPRITE
This command will shift the position of the drawn image without affecting the coordinate of the specified sprite. You can use this command to change the visible sprite in relation to the coordinates you use to position it. The parameters should be specified using integer values.

SYNTAX
```
OFFSET SPRITE Sprite Number, XOffset, YOffset
```

SCALE SPRITE
This command will expand or shrink the specified sprite according to the scale value provided. If the scale value is zero, the sprite will disappear. If the scale value is 100, the sprite will be set to its original size. If the scale value is set to 200, the size of the sprite will double. The parameters should be specified using integer values.

SYNTAX
```
SCALE SPRITE Sprite Number, Scale
```

SIZE SPRITE
This command will expand or shrink the specified sprite according to the size values provided. You must specify both a horizontal and vertical size when resizing sprites. The size values must be greater than zero of the command will fail. The parameters should be specified using integer values.

SYNTAX
```
SIZE SPRITE Sprite Number, XSize, YSize
```

STRETCH SPRITE
This command will expand or shrinks the specified sprite according to the scale values provided. You must specify both a horizontal and vertical scale when stretching sprites. If the scale value is zero, the sprite will disappear. If the scale value is 100, the sprite will be set to its original size. If the scale value is set to 200, the size of the sprite will double. The parameters should be specified using integer values.

SYNTAX
```
STRETCH SPRITE Sprite Number, XScale, YScale
```

ROTATE SPRITE
This command will rotate the specified sprite. You can rotate the sprite around 360 degrees specified using the Angle Value using a range of 0 to 359. The sprite number should be specified using an integer value. The angle should be specified using a real number.

SYNTAX

ROTATE SPRITE Sprite Number, Angle

FLIP SPRITE
This command will vertically flip the visible image of the specified sprite. The image itself is untouched, but the specified sprite will be drawn upside down. The parameter should be specified using an integer value.

SYNTAX

FLIP SPRITE Sprite Number

MIRROR SPRITE
This command will mirror the image of the sprite horizontally. The image itself is untouched, but the specified sprite will be drawn in reverse. The parameter should be specified using an integer value.

SYNTAX

MIRROR SPRITE Sprite Number

PASTE SPRITE
This command will paste the sprite image to the screen, at the specified coordinates. The sprite image pasted to the screen is identical to the current state of the sprite, taking into account scaling, flipping and mirroring. The parameters should be specified using integer values.

SYNTAX

PASTE SPRITE Sprite Number, XPos, YPos

CREATE ANIMATED SPRITE
This command will create an animated sprite. The command automatically builds a sequence of animation frames from an image file by cutting up the image into a grid specified by the Across and Down Values. The Image Number will be used to hold the complete image. Use the PLAY SPRITE command to see the individual frames of this image. The parameters should be specified using integer values.

SYNTAX

CREATE ANIMATED SPRITE Sprite Number, Filename, Across, Down, Image Number

PLAY SPRITE
This command will play an animated sprite. The command defines the start and end frames to be played. These frames must have been previously set up using the CREATE ANIMATED SPRITE command. The Delay Value specifies the delay factor between animating frames. A low value is fast, a high value is slow. The parameters should be specified using integer values.

SYNTAX

PLAY SPRITE Sprite Number, Start Frame, End Frame, Delay Value

SET SPRITE FRAME
This command will set the frame of the specified sprite. When a sprite contains an animated sequence of frames created with the CREATE ANIMATED SPRITE command, you can set the frame directly using this command. The parameters should be specified using integer values.

SYNTAX

SET SPRITE FRAME Sprite Number, Frame Value

SET SPRITE PRIORITY
This command will set the relative priority of the specified sprite. All sprites start with a value of zero giving them equal chance of being drawn last. By setting a single sprite a value of one will cause that sprite to be drawn last. You can specify a unique priority value for each sprite creating an order of drawing for every sprite in your program.

SYNTAX

SET SPRITE PRIORITY Sprite Number, Priority

SET SPRITE IMAGE
This command will set the image of the specified sprite. You can manipulate which image the sprite uses by specifying an existing Image for the sprite. The parameters should be specified using integer values.

SYNTAX
```
SET SPRITE IMAGE Sprite Number, Image Number
```

SET SPRITE ALPHA
This command will set the alpha value of the specified sprite. The alpha controls how much of the sprite is present on screen and is used to create a translucent sprite. The Alpha value range is 0 to 255, with 255 being completely solid and 0 being invisible. The parameters should be specified using integer values.

SYNTAX
```
SET SPRITE ALPHA Sprite Number, Alpha Value
```

SET SPRITE DIFFUSE
This command will set the diffuse values of the specified sprite. The diffuse values controls the quantity of colour the sprite uses from each RGB component. The value ranges are 0 to 255, with 255 being completely full and 0 being no color use. If you only specified a value in the Red component, the sprite would appear a shade of red. Same goes for the other colours, and combinations of colour. The parameters should be specified using integer values.

SYNTAX
```
SET SPRITE DIFFUSE Sprite Number, Red Value, Green Value, Blue Value
```

SET SPRITE TEXTURE COORD
This command will modify the internal UV data of the specified sprite. You can manipulate each vertex of the sprite by specifying a Vertex Index from 0 to 3. The UValue and VValue specify a real value typically from 0.0 to 1.0. The first two parameters should be specified using integer values.

SYNTAX
```
SET SPRITE TEXTURE COORD Sprite Number, Vertex Index, UValue, VValue
```

SPRITE EXIST
This command will return a one if the specified sprite exists, otherwise zero is returned. The parameter should be specified using an integer value.

SYNTAX
```
Return Integer=SPRITE EXIST(Sprite Number)
```

SPRITE X
This command will return an integer value of the current X position of the specified sprite. The parameter should be specified using an integer value.

SYNTAX
```
Return Integer=SPRITE X(Sprite Number)
```

SPRITE Y
This command will return an integer value of the current Y position of the specified sprite. The parameter should be specified using an integer value.

SYNTAX
```
Return Integer=SPRITE Y(Sprite Number)
```

SPRITE OFFSET X
This command will return an integer value of the current amount of X shift applied to the specified sprite. The parameter should be specified using an integer value.

SYNTAX

```
Return Integer=SPRITE OFFSET X(Sprite Number)
```

SPRITE OFFSET Y
This command will return an integer value of the current amount of Y shift applied to the specified sprite. The parameter should be specified using an integer value.

SYNTAX
```
Return Integer=SPRITE OFFSET Y(Sprite Number)
```

SPRITE SCALE X
This command will return an integer value of the specified sprite's horizontal scale. The parameter should be specified using an integer value.

SYNTAX
```
Return Integer=SPRITE SCALE X(Sprite Number)
```

SPRITE SCALE Y
This command will return an integer value of the specified sprite's vertical scale. The parameter should be specified using an integer value.

SYNTAX
```
Return Integer=SPRITE SCALE Y(Sprite Number)
```

SPRITE WIDTH
This command will return an integer value of the width of the specified sprite determined by the width of the current image being used. The parameter should be specified using an integer value.

SYNTAX
```
Return Integer=SPRITE WIDTH(Sprite Number)
```

SPRITE HEIGHT
This command will return an integer value of the height of the specified sprite determined by the height of the current image being used. The parameter should be specified using an integer value.

SYNTAX
```
Return Integer=SPRITE HEIGHT(Sprite Number)
```

SPRITE IMAGE
This command will return an integer value of the image number used by the specified sprite. The parameter should be specified using an integer value.

SYNTAX
```
Return Integer=SPRITE IMAGE(Sprite Number)
```

SPRITE MIRRORED
This command will return a one if the specified sprite has been mirrored horizontally, otherwise zero will be returned. The parameter should be specified using an integer value.

SYNTAX
```
Return Integer=SPRITE MIRRORED(Sprite Number)
```

SPRITE FLIPPED
This command will return a one if the specified sprite has been flipped vertically, otherwise zero will be returned. The parameter should be specified using an integer value.

SYNTAX
```
Return Integer=SPRITE FLIPPED(Sprite Number)
```

SPRITE HIT
This command will return a one if the specified sprite has impacted against the target sprite specified. If a target sprite has not been specified and a value of zero has been used, this command will return the sprite number of any sprite impacting against it. The parameters should be specified using integer values.

SYNTAX

Return Integer=SPRITE HIT(Sprite Number, Target Sprite Number)

SPRITE COLLISION
This command will return a one if the specified sprite is overlapping the target sprite specified. If a target sprite has not been specified and a value of zero has been used, this command will return the sprite number of any sprite overlapping it. The parameters should be specified using integer values.

SYNTAX

Return Integer=SPRITE COLLISION(Sprite Number, Target Sprite Number)

SPRITE ANGLE
This command will return the Angle of the specified sprite. The angle value determines how much the sprite is rotated, and ranges from 0 to 359. You should specify the Sprite Number as an integer value.

SYNTAX

Return Float=SPRITE ANGLE(Sprite Number)

SPRITE FRAME
This command will return the frame of the specified sprite. Frames are only returned from animated sprites created with the CREATE ANIMATED SPRITE command. You should specify the Sprite Number as an integer value.

SYNTAX

Return Integer=SPRITE FRAME(Sprite Number)

SPRITE ALPHA
This command will return the Alpha value of the specified sprite. The alpha value determines how much translucency the sprite has, 255 being completely solid. You should specify the Sprite Number as an integer value.

SYNTAX

Return Integer=SPRITE ALPHA(Sprite Number)

SPRITE RED
This command will return the diffuse Red value of the specified sprite. The value determines how much red is used in the sprite, 255 being completely used. You should specify the Sprite Number as an integer value.

SYNTAX

Return Integer=SPRITE RED(Sprite Number)

SPRITE GREEN
This command will return the diffuse Green value of the specified sprite. The value determines how much green is used in the sprite, 255 being completely used. You should specify the Sprite Number as an integer value.

SYNTAX

Return Integer=SPRITE GREEN(Sprite Number)

SPRITE BLUE
This command will return the diffuse Blue value of the specified sprite. The value determines how much blue is used in the sprite, 255 being completely used. You should specify the Sprite Number as an integer value.

SYNTAX

Return Integer=SPRITE BLUE(Sprite Number)

SPRITE VISIBLE
This command will return a one if the specified sprite is visible, otherwise zero is returned. The parameter should be specified using an integer value.

SYNTAX

```
Return Integer=SPRITE VISIBLE(Sprite Number)
```

IMAGE COMMAND SET

These commands provide functionality for the loading, display, modification and deleting of image graphics. You can load in a variety of picture file formats and position them anywhere on the screen to make your applications more visually appealing. In addition, images can be used as graphics for sprites and textures for 3D objects making them a very powerful medium.

LOAD IMAGE
This command will load a picture file as an image. The picture file must be of the BMP, JPG, TGA, DDS, DIB or PNG format. All images loaded are prepared with mipmaps if they do not already have them. TGA, DDS and PNG will retain their alpha channel data when loaded, providing the required transparency information when combined with transparency commands. A Texture Flag of zero will treat the image as a texture and load to fill a texture surface in memory. A value of one will preserve the image without adding mipmaps, scaling or filtering and so retaining its pixel perfect quality.

SYNTAX
```
LOAD IMAGE Filename, Image Number
LOAD IMAGE Filename, Image Number, Texture Flag
```

SAVE IMAGE
This command will save an image to a file. You must specify an existing image number between 1 and 65535. The file must not already exist, otherwise the command will fail. The image file format is determined by the extension given, and can be one of 'BMP', 'DDS', 'JPG' and 'DIB'.

SYNTAX
```
SAVE IMAGE Filename, Image Number
```

DELETE IMAGE
This command will delete the specified image from memory. You must not delete images that are being used by sprites, otherwise those sprites would disappear. Deleting unused images increases system performance. The parameter should be specified using an integer value.

SYNTAX
```
DELETE IMAGE Image Number
```

GET IMAGE
This command will copy a selected area of the current bitmap. Amongst other things, you can use this command to store sequences of image data and provide animations for sprites. When images are grabbed, they are stored in memory and do not require the bitmap from which the image was taken. The parameters should be specified using integer values. A texture flag of one will grab a pure image and not stretch and filter the image for texture usage. If the image is to be used as a texture, a texture flag of zero should be used.

SYNTAX
```
GET IMAGE Image Number, Left, Top, Right, Bottom
GET IMAGE Image Number, Left, Top, Right, Bottom, Texture Flag
```

PASTE IMAGE
This command will paste the specified image to the screen. Optionally, you can paste images to bitmaps using the SET CURRENT BITMAP command. If the optional transparent flag is set to one, all coloured pixels of RGB(0,0,0) are not drawn. The parameters should be specified using integer values.

SYNTAX
```
PASTE IMAGE Image Number, X, Y
PASTE IMAGE Image Number, X, Y, Transparency
```

SET IMAGE COLORKEY
This command will set the general colorkey for all images. A colorkey allows you to specify a particular RGB colour that will be treated as transparent by the image when used. Transparent pixels of an image are not drawn. You must use this

command before loading or grabbing an image in order that a suitable alpha map be generated from the transparent pixels of the desired image to be calculated. The alpha map is then used to decide which pixels are drawn and which pixels are never rendered.

SYNTAX

```
SET IMAGE COLORKEY Red Value, Green Value, Blue Value
```

IMAGE EXIST
This command will return a one if the image exists.

SYNTAX

```
Return Integer=IMAGE EXIST(Image Number)
```

ANIMATION COMMAND SET

These commands provide functionality for the loading, displaying and deleting of animation files. Animations can be loaded from common animation file formats such as MPEG and AVI, or direct from a DVD Disc if present on the system. Other animation formats are supported if your system is capable of loading and displaying them.

LOAD ANIMATION
This command loads an animation file into the specified animation number. The animation file must be of the MPEG, AVI, Quicktime, WAV, AIFF, AU or SND format. You must specify an Animation Number between 1 and 32. Once you have loaded the animation file successfully, you can use the specified animation number to place, play and stop the animation.

SYNTAX
```
LOAD ANIMATION Filename,Animation Number
```

LOAD DVD ANIMATION
This command will load a DVD movie if both a DVD driver exists on the system and a DVD CD is in a DVD Drive attached to the system. Once the DVD movie has been loaded, it can be played and controlled like any other animation. This command will fail silently. If a DVD does not exist to play, use the TOTAL DVD CHAPTERS() command to find out.

SYNTAX
```
LOAD DVD ANIMATION Animation Number
```

DELETE ANIMATION
This command deletes an animation previously loaded into the specified Animation Number. Deleting animations when you have finished with them improves system performance. If the animation is not stopped before the animation is deleted, the current frame of the animation remains on the screen or bitmap.

SYNTAX
```
DELETE ANIMATION Animation Number
```

PLAY ANIMATION
This command will play an animation on the screen or to the current Bitmap. By default, animations are played to the screen. You must provide an Animation Number of a previously loaded animation file. You can optionally provide either one or two sets of X and Y Coordinates to place and resize the animation anywhere on the screen.

SYNTAX
```
PLAY ANIMATION Animation Number
PLAY ANIMATION Animation Number,X,Y
PLAY ANIMATION Animation Number,X,Y,X,Y
PLAY ANIMATION Animation Number,Bitmap Number,X,Y,X,Y
```

PLAY ANIMATION TO IMAGE
This command will play a specified animation to an image. An image will be created, using the animation as a source of animating texture and copy the animation to the specified region within the image. You can use this as a rapid method of streaming an animation to any polygon.

SYNTAX
```
PLAY ANIMATION TO IMAGE Animation Number,Image,Left,Top,Right,Bottom
```

LOOP ANIMATION
This command plays the specified animation on the screen or to the current Bitmap, and repeats the animation continuously. You must provide an Animation Number of a previously loaded animation file.

SYNTAX
```
LOOP ANIMATION Animation Number
LOOP ANIMATION Animation Number,Bitmap Number,X,Y,X,Y
```

STOP ANIMATION
This command stops the specified animation if it is playing.

SYNTAX
```
STOP ANIMATION Animation Number
```

PAUSE ANIMATION
This command will pause the specified animation if it is playing.

SYNTAX
```
PAUSE ANIMATION Animation Number
```

RESUME ANIMATION
This command resumes the specified animation if it is currently paused.

SYNTAX
```
RESUME ANIMATION Animation Number
```

PLACE ANIMATION
This command redefines the drawing area of a previously loaded animation. Using this command, animations can be stretched, shrunk or moved across the screen even while the animation is playing.

SYNTAX
```
PLACE ANIMATION Animation Number,X,Y,X,Y
```

SET ANIMATION VOLUME
This command will set the volume of the specified animation. A value of 100 is normal, where 50 is half volume and 200 is double the volume. You can specify any integer value for the volume.

SYNTAX
```
SET ANIMATION VOLUME Animation Number, Volume
```

SET ANIMATION SPEED
This command will set the speed of the specified animation. A value of 100 is normal, where 50 is half speed and 200 is double the speed. You can specify any integer value for the speed.

SYNTAX
```
SET ANIMATION SPEED Animation Number, Speed
```

SET DVD CHAPTER
This command will set the chapter of a DVD and start it playing from the beginning. There can be between 1 and 99 titles per DVD volume, and between 1 and 999 chapters per title.

SYNTAX
```
SET DVD CHAPTER Animation Number, Title Number, Chapter Number
```

ANIMATION EXIST
This command will return a one if the specified animation exists, otherwise zero is returned.

SYNTAX
```
Return Integer=ANIMATION EXIST(Animation Number)
```

ANIMATION WIDTH
This command will return the current width of the specified animation. If you have resized the animation when playing or

placing then the width of the modified animation will be returned.

SYNTAX

Return Integer=ANIMATION WIDTH(Animation Number)

ANIMATION HEIGHT
This command will return the current height of the specified animation. If you have resized the animation when playing or placing then the height of the modified animation will be returned.

SYNTAX

Return Integer=ANIMATION HEIGHT(Animation Number)

ANIMATION PLAYING
This command will return a one if the specified animation is playing, otherwise zero is returned.

SYNTAX

Return Integer=ANIMATION PLAYING(Animation Number)

ANIMATION LOOPING
This command will return a one if the specified animation is looping, otherwise zero will be returned.

SYNTAX

Return Integer=ANIMATION LOOPING(Animation Number)

ANIMATION PAUSED
This command will return a one if the specified animation is paused, otherwise zero is returned..

SYNTAX

Return Integer=ANIMATION PAUSED(Animation Number)

ANIMATION POSITION X
This command will return the leftmost position of the specified animation via its X Coordinate.

SYNTAX

Return Integer=ANIMATION POSITION X(Animation Number)

ANIMATION POSITION Y
This command will return the topmost position of the specified animation via its Y Coordinate.

SYNTAX

Return Integer=ANIMATION POSITION Y(Animation Number)

ANIMATION VOLUME
This command will return the integer value representing the volume of the animation.

SYNTAX

Return Integer=ANIMATION VOLUME(Animation Number)

ANIMATION SPEED
This command will return the integer value representing the speed of the animation.

SYNTAX

Return Integer=ANIMATION SPEED(Animation Number)

TOTAL DVD CHAPTERS
This command will return the number of chapters contained in the specified title of the DVD volume. A value of zero means

the title value does not contain any chapters and can be treated as a non existent title.

SYNTAX

Return Integer=TOTAL DVD CHAPTERS(Animation Number, Title Number)

LIGHT COMMAND SET

These commands provide functionality to control 3D world space lights. Used in combination with the other 3D command sets, you can create, position, point, set and delete lights within your application. A light is responsible for illuminating a part of your 3D world. Lights look best when your 3D geometry uses many polygons, though there is a performance penalty based on the hardware your system is using.

MAKE LIGHT
This command will create a new light in the scene. You can create up to 7 new lights, numbered 1 to 7. Light zero is the default light and is always present. The light number must be specified using an integer value.

SYNTAX

```
MAKE LIGHT Light Number
```

DELETE LIGHT
This command will delete an existing light from the scene. Light zero is the default light and cannot be removed, only hidden. The light number must be specified using an integer value.

SYNTAX

```
DELETE LIGHT Light Number
```

SHOW LIGHT
This command will show an existing light within the scene. The light number must be specified using an integer value.

SYNTAX

```
SHOW LIGHT Light Number
```

HIDE LIGHT
This command will hide an existing light and remove its influence from the scene. The light number must be specified using an integer value.

SYNTAX

```
HIDE LIGHT Light Number
```

COLOR LIGHT
This command will color an existing light within the scene. You can specify the color as a single colourvalue using the RGB command. Alternatively you can specify the color values individually by providing a red, green and blue component value in the range of -255 to 255. The light number and parameters must be specified using integer values.

SYNTAX

```
COLOR LIGHT Light Number, Color Value
COLOR LIGHT Light Number, Red, Green, Blue
```

POSITION LIGHT
This command will position an existing light within the scene. Only spot lights and point lights can make use of this command. The light number must be specified using an integer value. The coordinates must be specified using real values.

SYNTAX

```
POSITION LIGHT Light Number, X, Y, Z
POSITION LIGHT Light Number, Vector
```

ROTATE LIGHT
This command will rotate an existing light within the scene. Only spot lights can make use of this command. The light number must be specified using an integer value. The angles must be specified using real values.

SYNTAX

ROTATE LIGHT Light Number, XAngle, YAngle, ZAngle

ROTATE LIGHT Light Number, Vector

POINT LIGHT
This command will point an existing light at a location within the scene. Only spot lights and directional lights can make use of this command. The light number must be specified using an integer value. The coordinates must be specified using real values.

SYNTAX

POINT LIGHT Light Number, X, Y, Z

SET DIRECTIONAL LIGHT
This command will set an existing light to that of a directional light. The direction is specified using directional values assuming that the origin of the light is 0,0,0. The light number must be specified using an integer value. The directional values must be specified using real values.

SYNTAX

SET DIRECTIONAL LIGHT Light Number, NX, NY, NZ

SET POINT LIGHT
This command will set an existing light to that of a point light. The position is specified using a coordinate in 3D space. The light number must be specified using an integer value. The coordinate must be specified using real values.

SYNTAX

SET POINT LIGHT Light Number, X, Y, Z

SET SPOT LIGHT
This command will set an existing light to that of a spot light. The spot light is defined by a constant cone of inner light and a gradual fading of light within an outer cone. The inner and outer cones are defined by an angle ranging from 0 to 360. The light number must be specified using an integer value. The angles must be specified using real values.

SYNTAX

SET SPOT LIGHT Light Number, Inner Angle, Outer Angle

SET LIGHT RANGE
This command will set the range of the light, which determines how far it reaches from its point of origin. Directional lights do not have a range.

SYNTAX

SET LIGHT RANGE Light Number, Distance

SET LIGHT TO OBJECT POSITION
This command will position an existing light to the location of another object. The parameters must be specified using integer values.

SYNTAX

SET LIGHT TO OBJECT POSITION Light Number, Object Number

SET LIGHT TO OBJECT ORIENTATION
This command will rotate an existing light to the orientation of another object. To create the effect of headlamps, two spotlights would each take the rotational orientation of a car. The parameters must be specified using integer values.

SYNTAX

SET LIGHT TO OBJECT ORIENTATION Light Number, Object Number

SET VECTOR3 TO LIGHT POSITION

This command will set the vector3 data using the X, Y and Z coordinates from the specified light position.

SYNTAX

```
SET VECTOR3 TO LIGHT POSITION Vector, Light Number
```

SET VECTOR3 TO LIGHT ROTATION
This command will set the vector3 data using the X, Y and Z angles from the specified light rotation.

SYNTAX

```
SET VECTOR3 TO LIGHT ROTATION Vector, Light Number
```

SET NORMALIZATION ON
This command will normalize all 'normals' contained in 3D rendering data.

SYNTAX

```
SET NORMALIZATION ON
```

SET NORMALIZATION OFF
This command will deactivate normalization of all 'normals' data.

SYNTAX

```
SET NORMALIZATION OFF
```

FOG ON
This command will activate the effect of fogging, if the current display card supports it. All fog color and distance settings are safely restored when using this command.

SYNTAX

```
FOG ON
```

FOG OFF
This command will deactivate the effect of fogging, if the fog has been previously activated using the FOG ON command. All fog color and distance settings are safely stored when using this command, allowing a call to FOG ON to restore all fog settings.

SYNTAX

```
FOG OFF
```

FOG COLOR
This command will set the color of the fogging effect. The parameter should be specified using an integer value. The color value can be generated using the RGB command to create over 16 million different colors of fog.

SYNTAX

```
FOG COLOR Color Value
FOG COLOR Red, Green, Blue
```

FOG DISTANCE
This command will set the visible distance of the fog based on the view from the camera. The distance value represents the Z depth at which the fog obscures 3D objects. A distance of zero sets the fog to obscure the camera entirely. A distance of 5000 places the fog to obscure 3D objects as they are Z clipped by the system. A distance greater than 5000 will not obscure distant 3D objects and will allow the objects to be visibly clipped. The parameters should be specified using integer values. The parameter should be specified using an integer value.

SYNTAX

```
FOG DISTANCE Distance
```

SET AMBIENT LIGHT

This command will set the percentage level of ambient light. A setting of 100 provides full illumination and no shadow whereas a setting of zero gives no illumination and substantial shadowing on any 3D object. The parameter should be specified using an integer value.

SYNTAX

```
SET AMBIENT LIGHT Percentage
```

COLOR AMBIENT LIGHT
This command will changes the current ambient light colour.

SYNTAX

```
COLOR AMBIENT LIGHT Color Value
```

LIGHT EXIST
This command will return a one of the light exists.

SYNTAX

```
Return Integer=LIGHT EXIST(Light Number)
```

LIGHT TYPE
This command will return 1 for point, 2 for spot and 3 for directional light type.

SYNTAX

```
Return Integer=LIGHT TYPE(Light Number)
```

LIGHT VISIBLE
This command will return a 1 if the light is active.

SYNTAX

```
Return Integer=LIGHT VISIBLE(Light Number)
```

LIGHT RANGE
This command will return the current range of the light source.

SYNTAX

```
Return Float=LIGHT RANGE(Light Number)
```

LIGHT POSITION X
This command will return the X position of the light .

SYNTAX

```
Return Float=LIGHT POSITION X(Light Number)
```

LIGHT POSITION Y
This command will return the Y position of the light.

SYNTAX

```
Return Float=LIGHT POSITION Y(Light Number)
```

LIGHT POSITION Z
This command will return the Z position of the light.

SYNTAX

```
Return Float=LIGHT POSITION Z(Light Number)
```

LIGHT DIRECTION X

This command will return the X value of the light direction.

SYNTAX

Return Float=LIGHT DIRECTION X(Light Number)

LIGHT DIRECTION Y

This command will return the Y value of the light direction.

SYNTAX

Return Float=LIGHT DIRECTION Y(Light Number)

LIGHT DIRECTION Z

This command will return the Z value of the light direction.

SYNTAX

Return Float=LIGHT DIRECTION Z(Light Number)

CAMERA COMMAND SET

These commands provide functionality to control 3D world space cameras. Used in combination with the other 3D command sets, you can create, position, point, set and delete cameras within your application. A camera is responsible for deciding which part of your 3D world is drawn to the screen. With multiple cameras you can draw different parts of your world from different positions on screen, or to an offscreen bitmap and even to a 3D object.

MAKE CAMERA

This command will create a new camera for the 3D scene. You can position this camera anywhere in the scene and alter the output view of this camera using the SET CAMERA VIEW command.

SYNTAX

```
MAKE CAMERA Camera Number
```

DELETE CAMERA

This command will delete a current camera from the scene. You must specify an existing camera number using an integer value.

SYNTAX

```
DELETE CAMERA Camera Number
```

MOVE CAMERA

This command will move the camera in the direction it is facing. The step value specifies how far to move the camera and should be a real number.

SYNTAX

```
MOVE CAMERA Distance Value
MOVE CAMERA Camera Number, Distance Value
```

POSITION CAMERA

This command will set the position of the camera in 3D space. The coordinates should be real numbers.

SYNTAX

```
POSITION CAMERA X, Y, Z
POSITION CAMERA Camera Number, X, Y, Z
POSITION CAMERA Camera Number, Vector
```

POINT CAMERA

This command will point the camera to a point in 3D space. The coordinates should be real numbers.

SYNTAX

```
POINT CAMERA X, Y, Z
POINT CAMERA Camera Number, X, Y, Z
```

ROTATE CAMERA

This command will rotate the camera around its X, Y and Z axis. The angle values should be real numbers.

SYNTAX

```
ROTATE CAMERA XAngle, YAngle, ZAngle
ROTATE CAMERA Camera Number, XAngle, YAngle, ZAngle
ROTATE CAMERA Camera Number, Vector
```

XROTATE CAMERA

This command will rotate the camera around its X axis. The angle value should be a real number.

SYNTAX
```
XROTATE CAMERA XAngle
XROTATE CAMERA Camera Number, XAngle
```

YROTATE CAMERA
This command will rotate the camera around its Y axis. The angle value should be a real number.

SYNTAX
```
YROTATE CAMERA YAngle
YROTATE CAMERA Camera Number, YAngle
```

ZROTATE CAMERA
This command will rotate the camera around its Z axis. The angle value should be a real number.

SYNTAX
```
ZROTATE CAMERA ZAngle
ZROTATE CAMERA Camera Number, ZAngle
```

TURN CAMERA LEFT
This command will turn the camera left. The rotation is independent of any axis orientation and allows free motion. The value of the angle can be positive or negative. The angle must be specified using a real value.

SYNTAX
```
TURN CAMERA LEFT Angle Value
TURN CAMERA LEFT Camera Number, Angle Value
```

TURN CAMERA RIGHT
This command will turn the camera right. The rotation is independent of any axis orientation and allows free motion. The value of the angle can be positive or negative. The angle must be specified using a real value.

SYNTAX
```
TURN CAMERA RIGHT Angle Value
TURN CAMERA RIGHT Camera Number, Angle Value
```

PITCH CAMERA UP
This command will pitch the camera upwards. The rotation is independent of any axis orientation and allows free motion. The value of the angle can be positive or negative. The angle must be specified using a real value.

SYNTAX
```
PITCH CAMERA UP Angle Value
PITCH CAMERA UP Camera Number, Angle Value
```

PITCH CAMERA DOWN
This command will pitch the camera downwards. The rotation is independent of any axis orientation and allows free motion. The value of the angle can be positive or negative. The angle must be specified using a real value.

SYNTAX
```
PITCH CAMERA DOWN Angle Value
PITCH CAMERA DOWN Camera Number, Angle Value
```

ROLL CAMERA LEFT
This command will roll the camera left. The rotation is independent of any axis orientation and allows free motion. The value of the angle can be positive or negative. The angle must be specified using a real value.

SYNTAX

```
ROLL CAMERA LEFT Angle Value
ROLL CAMERA LEFT Camera Number, Angle Value
```

ROLL CAMERA RIGHT
This command will roll the camera right. The rotation is independent of any axis orientation and allows free motion. The value of the angle can be positive or negative. The angle must be specified using a real value.

SYNTAX

```
ROLL CAMERA RIGHT Angle Value
ROLL CAMERA RIGHT Camera Number, Angle Value
```

SET CURRENT CAMERA
This command will select the current active camera. When more than one camera exists, and a camera command is used without specifying a camera number, the active camera is used.

SYNTAX

```
SET CURRENT CAMERA Camera Number
```

CLEAR CAMERA VIEW
This command will clear the viewport of the camera. The viewport is the actual area on screen where all 3D is drawn. The default viewport area is the entire screen. You can specify that only the camera viewport be cleared using this command. This area can be changed using the Set Camera View command. You must specify the colour value using an integer value.

SYNTAX

```
CLEAR CAMERA VIEW Color Value
CLEAR CAMERA VIEW Camera Number, Color Value
```

SET CAMERA FOV
This command will set field of view for the camera given an Angle value. The default angle is the result of the calculation 360 degrees divided by four.

SYNTAX

```
SET CAMERA FOV Field-Of-View Angle
SET CAMERA FOV Camera Number, Field-Of-View Angle
```

SET CAMERA RANGE
This command will set the viewing range of the camera. The Front Value specifies the closest point beyond which the camera starts to draw the 3D scene. The Back Value specifies the furthest point beyond which the camera stops drawing the 3D scene. The parameters must be specified using real numbers greater than zero. The default range starts drawing the 3D scene with a front value of 1 and a back value of 3000.

SYNTAX

```
SET CAMERA RANGE Near Value, Far Value
SET CAMERA RANGE Camera Number, Near Value, Far Value
```

SET CAMERA VIEW
This command will set the viewport of the camera. The viewport is the actual area on screen where all 3D is drawn. The default viewport area is the entire screen, but can be specified using this command. You must specify the screen coordinates using integer values.

SYNTAX

```
SET CAMERA VIEW Left, Top, Right, Bottom
SET CAMERA VIEW Camera Number, Left, Top, Right, Bottom
```

SET CAMERA ASPECT
This command will adjust the aspect ratio at which the camera renders to the screen. By default this aspect ratio is calculated by dividing the screen width by the screen height, normally resulting in an aspect ratio of 0.75. You can change this aspect ratio using this command in situations where you would like to render a perfect square to the screen or if you are writing for hardware that spans a single surface over multiple display devices.

SYNTAX

```
SET CAMERA ASPECT Aspect Ratio
SET CAMERA ASPECT Camera Number, Aspect Ratio
```

SET CAMERA ROTATION XYZ
This command will reverse the camera rotation order. The cameras normal rotation order is to first rotate on the Z axis, then on the Y axis and finally on the X axis. This command reverses this order to XYZ.

SYNTAX

```
SET CAMERA ROTATION XYZ
SET CAMERA ROTATION XYZ Camera Number
```

SET CAMERA ROTATION ZYX
This command will restore the default camera rotation order. The cameras normal rotation order is to first rotate on the Z axis, then on the Y axis and finally on the X axis. This command restores the order to ZYX.

SYNTAX

```
SET CAMERA ROTATION ZYX
SET CAMERA ROTATION ZYX Camera Number
```

SET CAMERA TO FOLLOW
This command automatically controls the camera to provide a tracking system. By providing the 3D world coordinate of the entity you wish to track, and some camera data, your camera will automatically update its current position each time this command is called. The X, Y, Z and Angle values provide the coordinate to track. The Distance value specifies the required distance between the coordinate and the camera. The Height value specifies the required height of the camera in 3D space. The Smooth value specifies the level of smoothing required for the camera, where a value of 1.0 is no smoothing and a value of 100.0 is lots of smoothing. The Collision value is a special flag that allows the camera to detect whether it is hitting any of the static collision boxes and if set to one will automatically adjust so as not to enter these collidable areas.

SYNTAX

```
SET CAMERA TO FOLLOW X, Y, Z, Angle, Distance, Height, Smooth, Collision
SET CAMERA TO FOLLOW Camera Number, X, Y, Z, Angle, Distance, Height, Smooth, Collision
```

SET CAMERA TO IMAGE
This command will direct all camera output to the specified image. This command is ideal for performing the fastest method of creating textured polygons that show a different view within your 3D world. You can use this command to create mirrors, security cameras within your game or dynamically moving the camera view around on a polygon shape such as a floating panel.

SYNTAX

```
SET CAMERA TO IMAGE Camera Number, Image Number, Width, Height
```

SET CAMERA TO OBJECT ORIENTATION
This command will set the camera to the same direction as the specified 3D object. The object number must be an integer value.

SYNTAX

```
SET CAMERA TO OBJECT ORIENTATION Object Number
SET CAMERA TO OBJECT ORIENTATION Camera Number, Object Number
```

SET VECTOR3 TO CAMERA POSITION
This command will set the vector3 data using the X, Y and Z coordinates from the specified camera position.

SYNTAX

```
SET VECTOR3 TO CAMERA POSITION Vector, Camera Number
```

SET VECTOR3 TO CAMERA ROTATION
This command will set the vector3 data using the X, Y and Z angles from the specified camera rotation.

SYNTAX

```
SET VECTOR3 TO CAMERA ROTATION Vector, Camera Number
```

CONTROL CAMERA USING ARROWKEYS
This command will monitor the arrow keys and move the camera based on their activity. The up and down arrow keys move the camera forward and backward. The left and right arrow keys turn the camera left and right respectively.

SYNTAX

```
CONTROL CAMERA USING ARROWKEYS Camera Number, MoveSpeed, TurnSpeed
```

AUTOCAM ON
This command will activate the auto camera which will reposition when a new object is loaded or created.

SYNTAX

```
AUTOCAM ON
```

AUTOCAM OFF
This command will deactivate the auto camera. The camera will then no longer reposition when a new object is loaded or created.

SYNTAX

```
AUTOCAM OFF
```

BACKDROP ON
This command will activate a 3D backdrop that fills the visible screen. The backdrop is automatically activated the first time any 3D object is created or loaded in order to clear the background screen. The backdrop can also be colored, textured and scrolled to create the effects of sky or other background effects. If you wish to set-up the backdrop before creating your objects, use this command to activate it.

SYNTAX

```
BACKDROP ON
BACKDROP ON Camera Number
```

BACKDROP OFF
This command will deactivate the 3D backdrop preventing it from being drawn to the screen. The backdrop is automatically activated the first time any 3D object is created or loaded in order to clear the background screen. If you do not wish the backdrop to automatically activate, use this command at the start of your program.

SYNTAX

```
BACKDROP OFF
BACKDROP OFF Camera Number
```

COLOR BACKDROP
This command will COLOR the 3D backdrop in the specified COLOR. You can specify the COLOR of your choice by using the RGB command to generate the COLOR value to pass into the command. The parameter should be specified using an integer value.

SYNTAX

```
COLOR BACKDROP Color Value
COLOR BACKDROP Camera Number, Color Value
```

TEXTURE BACKDROP
This command will texture the 3D backdrop using the specified image value. This command is now obsolete and using sky spheres and boxes is a recommended alternative.

SYNTAX
```
TEXTURE BACKDROP Image Number
TEXTURE BACKDROP Camera Number, Image Number
```

SCROLL BACKDROP
This command will scroll the 3D backdrop using the specified X and Y scroll values. This command is now obsolete and using sky spheres and boxes is a recommended alternative.

SYNTAX
```
SCROLL BACKDROP U, V
SCROLL BACKDROP Camera Number, U, V
```

CAMERA POSITION X
This command will return the real value X position of the camera in 3D space.

SYNTAX
```
Return Float=CAMERA POSITION X()
Return Float=CAMERA POSITION X(Camera Number)
```

CAMERA POSITION Y
This command will return the real value Y position of the camera in 3D space.

SYNTAX
```
Return Float=CAMERA POSITION Y()
Return Float=CAMERA POSITION Y(Camera Number)
```

CAMERA POSITION Z
This command will return the real value Z position of the camera in 3D space.

SYNTAX
```
Return Float=CAMERA POSITION Z()
Return Float=CAMERA POSITION Z(Camera Number)
```

CAMERA ANGLE X
This command will return the real value X angle of the camera.

SYNTAX
```
Return Float=CAMERA ANGLE X()
Return Float=CAMERA ANGLE X(Camera Number)
```

CAMERA ANGLE Y
This command will return the real value Y angle of the camera.

SYNTAX
```
Return Float=CAMERA ANGLE Y()
Return Float=CAMERA ANGLE Y(Camera Number)
```

CAMERA ANGLE Z
This command will return the real value Z angle of the camera.

SYNTAX

Return Float=CAMERA ANGLE Z()

Return Float=CAMERA ANGLE Z(Camera Number)

BASIC3D COMMAND SET

These commands provide fundamental access to the loading, creation, modification and deleting of 3D geometry. Due to the size of the category, you will find a section header that groups common commands together. With this command set you have control over loading many 3D file formats, powerful texturing commands, sub-object manipulation, collision and an array of expressions that return data on every facet of your systems 3D capabilities.

OBJECT COMMANDS

LOAD OBJECT
This command loads a model into the specified 3D object number. You must specify a model in the X, 3DS, MDL, MD2 or MD3 format. Once you have loaded the 3D object file successfully, you can use the specified 3D object number to position, rotate, scale, animate and manipulate your 3D object. The object number should be specified using an integer value.

SYNTAX

```
LOAD OBJECT Filename, Object Number
```

APPEND OBJECT
This command will append all animation data from an X file into the specified X file object. The new animation data will begin from the start frame specified up to the length of the files animation data. Ensure that the 3D object being appended uses the same limb names as the original object, otherwise the load will fail. The parameters should be specified using integer values.

SYNTAX

```
APPEND OBJECT Filename, Object Number, Start Frame
```

CLONE OBJECT
This command will clone the specified object and make an exact duplicate of it. Cloned objects do not share any data allowing you to use the new object in any manner you wish.

SYNTAX

```
CLONE OBJECT Object Number, Source Object
```

INSTANCE OBJECT
This command will create an instance copy of the object. Unlike CLONE OBJECT, this instance will share most of the original objects data and be dependent on the original object remaining in existence to hold such data. Instanced objects do have the ability to hide/show both their limbs and hide/show the object, independent of the original object. This command is ideally suited to copying large numbers of primarily static objects such as trees and rocks where the original model data is to remaing largely unchanged.

SYNTAX

```
INSTANCE OBJECT Object Number, Source Object
```

DELETE OBJECT
This command will delete the specified 3D object previously loaded. The parameter should be specified using an integer value.

SYNTAX

```
DELETE OBJECT Object Number
```

SHOW OBJECT
This command will reveal a specified 3D object that was previously hidden. The parameter should be specified using an integer value.

SYNTAX

```
SHOW OBJECT Object Number
```

HIDE OBJECT
This command will hide the specified 3D object from view. You can substantially increase the performance of your 3D program if you hide objects whenever possible. The contents of a room behind a closed door can be hidden for as long as the door remains closed, allowing your program to run much faster and improve overall performance. The parameter should be specified using an integer value.

SYNTAX

```
HIDE OBJECT Object Number
```

POSITION OBJECT
This command will place the specified 3D object in 3D space. In order to see your 3D object, you must ensure the camera is pointing in the right direction and that both camera and 3D object are within 5000 units from each other. The object number should be specified using an integer value. The 3D Coordinates should be specified using real numbers.

SYNTAX

```
POSITION OBJECT Object Number, X, Y, Z
```

SCALE OBJECT
This command will scale the specified 3D object to stretch or shrink in all three dimensions, using percentage scale values. The object number should be specified using an integer value and the scale specified using real values.

SYNTAX

```
SCALE OBJECT Object Number, XSize, YSize, ZSize
```

ROTATE OBJECT
This command will rotate the specified 3D object around all three dimensions. The object number should be specified using an integer value. This method of rotation is called euler rotation and differs from free flight rotation. Euler angles retrieved from a free flight rotation specify a ZYX rotation, and you must call the SET OBJECT ROTATION ZYX command on the Euler based object to make the two rotation systems compatible. The rotation angles should be specified using real numbers.

SYNTAX

```
ROTATE OBJECT Object Number, XAngle, YAngle, ZAngle
```

MOVE OBJECT
This command will move the specified 3D object in 3D space. The command uses the current direction of the object and moves it using the specified step value. In order to see your 3D object, you must ensure the camera is pointing in the right direction and that both camera and 3D object are within 5000 units from each other. The object number should be specified using an integer value. The step value should be specified using a real number.

SYNTAX

```
MOVE OBJECT Object Number, Speed
```

POINT OBJECT
This command will point the specified 3D object towards a point in 3D space. The command sets the current direction of the object to face towards this point in space. The object number should be specified using an integer value. The 3D Coordinates should be specified using real numbers.

SYNTAX

```
POINT OBJECT Object Number, X, Y, Z
```

MOVE OBJECT DOWN
This command will move the object in a down direction relative to its forward facing angle, rather than an absolute world direction.

SYNTAX

```
MOVE OBJECT DOWN Object Number, Value
```

MOVE OBJECT LEFT

This command will move the object in a left direction relative to its forward facing angle, rather than an absolute world direction.

SYNTAX

```
MOVE OBJECT LEFT Object Number, Value
```

MOVE OBJECT RIGHT
This command will move the object in a right direction relative to its forward facing angle, rather than an absolute world direction.

SYNTAX

```
MOVE OBJECT RIGHT Object Number, Value
```

MOVE OBJECT UP
This command will move the object in a up direction relative to its forward facing angle, rather than an absolute world direction.

SYNTAX

```
MOVE OBJECT UP Object Number, Value
```

XROTATE OBJECT
This command will rotate the specified 3D object around the X axis dimension. The object number should be specified using an integer value. The rotation angle should be specified using a real number.

SYNTAX

```
XROTATE OBJECT Object Number, XAngle
```

YROTATE OBJECT
This command will rotate the specified 3D object around the Y axis dimension. The object number should be specified using an integer value. The rotation angle should be specified using a real number.

SYNTAX

```
YROTATE OBJECT Object Number, YAngle
```

ZROTATE OBJECT
This command will rotate the specified 3D object around the Z axis dimension. The object number should be specified using an integer value. The rotation angle should be specified using a real number.

SYNTAX

```
ZROTATE OBJECT Object Number, ZAngle
```

TURN OBJECT LEFT
This command will rotate an existing 3D object to turn left. The rotation is independent of any axis orientation and allows free motion. The value of the angle can be positive or negative. The object number must be specified using an integer value. The angle must be specified using a real value.

SYNTAX

```
TURN OBJECT LEFT Object Number, Value
```

TURN OBJECT RIGHT
This command will rotate an existing 3D object to turn right. The rotation is independent of any axis orientation and allows free motion. The value of the angle can be positive or negative. The object number must be specified using an integer value. The angle must be specified using a real value.

SYNTAX

```
TURN OBJECT RIGHT Object Number, Value
```

PITCH OBJECT UP

This command will rotate an existing 3D object to pitch upwards. The rotation is independent of any axis orientation and allows free motion. The value of the angle can be positive or negative. The object number must be specified using an integer value. The angle must be specified using a real value.

SYNTAX

```
PITCH OBJECT UP Object Number, Value
```

PITCH OBJECT DOWN
This command will rotate an existing 3D object to pitch downwards. The rotation is independent of any axis orientation and allows free motion. The value of the angle can be positive or negative. The object number must be specified using an integer value. The angle must be specified using a real value.

SYNTAX

```
PITCH OBJECT DOWN Object Number, Value
```

ROLL OBJECT LEFT
This command will rotate an existing 3D object to roll left. The rotation is independent of any axis orientation and allows free motion. The value of the angle can be positive or negative. The object number must be specified using an integer value. The angle must be specified using a real value.

SYNTAX

```
ROLL OBJECT LEFT Object Number, Value
```

ROLL OBJECT RIGHT
This command will rotate an existing 3D object to roll right. The rotation is independent of any axis orientation and allows free motion. The value of the angle can be positive or negative. The object number must be specified using an integer value. The angle must be specified using a real value.

SYNTAX

```
ROLL OBJECT RIGHT Object Number, Value
```

PLAY OBJECT
These commands will play the animation data contained within the specified 3D object. You can optionally play the animation by providing a specified start and end frame. The parameters should be specified using integer values.

SYNTAX

```
PLAY OBJECT Object Number
PLAY OBJECT Object Number, Start Frame
PLAY OBJECT Object Number, Start Frame, End Frame
```

LOOP OBJECT
This command will play and loop the animation data contained within the specified 3D object from the beginning. You can optionally play and loop the animation by providing a specified start and end frame. The parameters should be specified using integer values.

SYNTAX

```
LOOP OBJECT Object Number
LOOP OBJECT Object Number, Start Frame
LOOP OBJECT Object Number, Start Frame, End Frame
```

STOP OBJECT
This command will stop the animation in a specified 3D object. The parameter should be specified using an integer value.

SYNTAX

```
STOP OBJECT Object Number
```

SET OBJECT FRAME

This command will set the animation frame of the specified 3D object. The object number should be specified using an integer value. The frame number should be specified using a real number.

SYNTAX

```
SET OBJECT FRAME Object Number, Frame
```

SET OBJECT SPEED
This command will set the speed of the animation in the specified 3D object. A value of 1 will perform the animation at its slowest rate. A value of 100 is the maximum speed setting. The parameters should be specified using integer values.

SYNTAX

```
SET OBJECT SPEED Object Number, Speed
```

SET OBJECT INTERPOLATION
This command will set the percentage of animation frame interpolation in the specified 3D object. Interpolation occurs when the animation frame is manually set using the SET OBJECT FRAME command. If the interpolation percentage is 100, the transition between frames is instant. When the value is set to 50, it would take two cycles to make the transition. You can use interpolation to make extremely smooth movements between two different animation frames. The parameters should be specified using integer values.

SYNTAX

```
SET OBJECT INTERPOLATION Object Number, Interpolation
```

CONSTRUCTION COMMANDS

MAKE OBJECT
This command will construct a 3D object from a single mesh and image. The mesh is used as the root limb for the 3D object and the image is used as a texture for the object. You do not have to specify an image value, but such models will appear white when displayed. The parameters should be specified using integer values.

SYNTAX

```
MAKE OBJECT Object Number, Mesh Index, Image Number
```

MAKE OBJECT BOX
This command will construct a 3D object from a box mesh. The mesh is used as the root limb for the 3D object. The 3D object will be constructed untextured and such models will appear white when displayed. The parameters should be specified using integer values.

SYNTAX

```
MAKE OBJECT BOX Object Number, Width, Height, Depth
```

MAKE OBJECT CONE
This command will construct a 3D object from a cone mesh. The mesh is used as the root limb for the 3D object. The 3D object will be constructed untextured and such models will appear white when displayed. The parameters should be specified using integer values.

SYNTAX

```
MAKE OBJECT CONE Object Number, Size
```

MAKE OBJECT CUBE
This command will construct a 3D object from a cube mesh. The mesh is used as the root limb for the 3D object. The 3D object will be constructed untextured and such models will appear white when displayed. The parameters should be specified using integer values.

SYNTAX

```
MAKE OBJECT CUBE Object Number, Size
```

MAKE OBJECT CYLINDER
This command will construct a 3D object from a cylinder mesh. The mesh is used as the root limb for the 3D object. The 3D

object will be constructed untextured and such models will appear white when displayed. The parameters should be specified using integer values.

SYNTAX

```
MAKE OBJECT CYLINDER Object Number, Size
```

MAKE OBJECT FROM LIMB
This command will make a simple object from the specified limb. You can use this command to remove a limb element such as an arm or gun from an object and create a new object using just this element. Ideal for deconstructing loaded models and using their elements for other uses.

SYNTAX

```
MAKE OBJECT FROM LIMB Object Number, Second Object, Limb Number
```

MAKE OBJECT PLAIN
This command will construct a 3D object from a single sided flat mesh. The mesh is used as the root limb for the 3D object. The 3D object will be constructed untextured and such models will appear white when displayed. The parameters should be specified using integer values. By default this primitive is facing away from the camera when created and so any lighting is being performed on the side not immediately visible to the camera. This causes such properties such as shading and colouring not to be seen until the primitive is rotated to face the camera.

SYNTAX

```
MAKE OBJECT PLAIN Object Number, Width, Height
```

MAKE OBJECT SPHERE
This command will construct a 3D object from a sphere mesh. The mesh is used as the root limb for the 3D object. The 3D object will be constructed untextured and such models will appear white when displayed. The parameters should be specified using integer values.

SYNTAX

```
MAKE OBJECT SPHERE Object Number, Size
MAKE OBJECT SPHERE Object Number, Size, Rows, Columns
```

MAKE OBJECT TRIANGLE
This command will construct a 3D object from values that describe a single triangle mesh. The mesh is used as the root limb for the 3D object. The 3D object will be constructed untextured and such models will appear white when displayed. The object number should be specified using an integer value and the 3D coordinates specified using real values.

SYNTAX

```
MAKE OBJECT TRIANGLE Object Number, X1, Y1, Z1, X2, Y2, Z2, X3, Y3, Z3
```

APPEARANCE COMMANDS

TEXTURE OBJECT
This command will texture an object using the specified image. The image can be any size, but the quality of the texture depends on the graphics card used. A texture size to the power of two is recommended size for all your textures. Where an image is larger than the supported texture size, the image is internally reduced when used as a texture. The object and image number must be integer values. You can use this command to remove the effects of multitexturing effects and shaders. An additional texture stage index can be provided to the command to specify multi-textures directly, and is required when using shaders that take pixel data from secondary textures.

SYNTAX

```
TEXTURE OBJECT Object Number, Image Number
TEXTURE OBJECT Object Number, Stage Index, Image Number
```

COLOR OBJECT
This command will color the specified 3D object using an RGB colour value. The colour value can be specified using the RGB() command. The parameters should be specified using integer values. Some objects loaded from a model file may not be colored if they do not already contain diffuse color data.

SYNTAX

COLOR OBJECT Object Number, Color Value

GHOST OBJECT ON
This command will make the specified 3D object semi-transparent if supported by the current display card. This technique is known as alpha-blending and causes the object to appear as a ghost image. If a value of one is specified for the dark ghosting, the object uses negative alphablending. The parameter should be specified using an integer value.

SYNTAX

GHOST OBJECT ON Object Number

GHOST OBJECT ON Object Number, Ghost Effect

GHOST OBJECT OFF
This command will deactivate the effect of ghosting on the specified 3D object. The parameter should be specified using an integer value.

SYNTAX

GHOST OBJECT OFF Object Number

FADE OBJECT
This command will make the specified 3D object fade to the current ambient light level. With ambient light set to zero and the object faded using a value of zero, the object will be completely unlit. With a fade value of 200 its illumination will be doubled. With a fade value of 50, the illumination is halved. This technique can also be used with a ghosted object to slowly fade an object until completely invisible. The parameter should be specified using an integer value.

SYNTAX

FADE OBJECT Object Number, Fade Value

LOCK OBJECT ON
This command will lock the specified 3D object to the screen. Locking objects to the screen commands the object to completely ignore the camera's influence. A locked object will be positioned as though the camera had never been altered from its default orientation. To make locked objects visible, simply set the Z position to a significant positive value. The Object Number should be specified using an integer value.

SYNTAX

LOCK OBJECT ON Object Number

LOCK OBJECT OFF
This command will unlock the specified 3D object from the screen. Locking objects to the screen commands the object to completely ignore the camera's influence. By unlocking a previously locked object, you are commanding the object to associate itself with the influence of the camera once again. The Object Number should be specified using an integer value.

SYNTAX

LOCK OBJECT OFF Object Number

SET OBJECT TEXTURE
This command will set different texture modes used by the specified object. Every texture is painted onto an object using an internal set of values called UV data. This data contains a range of real numbers from zero to one. Zero specifying the top/left corner of your texture and one being the bottom/right corner of your texture. When an object uses UV data greater and less than this range, you are permitted a number of texture wrap modes to describe what should happen to paint these areas. Setting the Texture Wrap Mode to zero will use the default wrap mode which repeat the pattern of the texture over and over, a mode of one will mirror the texture to create a seamless texture pattern and a mode of two will set clamping which retains the colour of the last pixel at the textures edge and paint with that throughout the out of range area. The Mipmap Generation Flag is used to ensure the image has a mipmap texture. A mipmap is a texture that has many levels of detail, which the object can select and use based on the objects distance from the camera. Use integer values to specify the parameters.

SYNTAX

SET OBJECT TEXTURE Object Number, Texturing Mode, Mipmap Flag

SCALE OBJECT TEXTURE
This command will scale the UV data of the specified object. The UV data controls how a texture is mapped onto your object. By scaling the UV data, you can effectively stretch or tile the texture over your object. The U value controls the horizontal spread of the data. The V value controls the vertical spread of the data. A U or V value of 1 means no scale change. A value of 0.5 will scale the texture by half. A value of 2.0 will double the scale of the texture. The scale effect is permanent.

SYNTAX

```
SCALE OBJECT TEXTURE Object Number, UScale, VScale
```

SCROLL OBJECT TEXTURE
This command will scroll the UV data of the specified object. The UV data controls how a texture is mapped onto your object. By scrolling the UV data, you can effectively scroll the texture over your object. The U value controls the horizontal shift of the data. The V value controls the vertical shift of the data. The scroll effect is permanent.

SYNTAX

```
SCROLL OBJECT TEXTURE Object Number, X, Y
```

SET TEXTURE MD3
This command will set the specified object to use a special set of textures. The object must have been previously loaded as an MD3 model. MD3 models are blank by default and require this command to provide the textures. The six texture parameters H0, H1, L0, L1, L2 and U0 are values to images you must load prior to calling this command.

SYNTAX

```
SET TEXTURE MD3 Object Number, H0, H1, L0, L1, L2, U0
```

SET OBJECT SMOOTHING
This command will smooth the sharp edges of the mesh within an object by adjusting the normals data. A percentage value of zero will perform no smoothing and create a facet surface for each limb within the object. A percentage value of 100 will perform full smoothing, averaging all normals that share a vertex position and create a smoothing effect elimiating all edges. A value between these two limits will determine the degree beyond which an edge will be smoothed or left sharp.

SYNTAX

```
SET OBJECT SMOOTHING Object Number, Percentage
```

SHOW OBJECT BOUNDS
This command will show the wireframe bounds that reveal the collision sphere and collision boundbox associated with the object. These bounds can later be hidden by using the HIDE OBJECT BOUNDS command. By specifying the optional Box Only Flag, the bound sphere is not shown with the box, allowing a clearer view of the bounding box on the object.

SYNTAX

```
SHOW OBJECT BOUNDS Object Number
SHOW OBJECT BOUNDS Object Number, Box Only Flag
```

HIDE OBJECT BOUNDS
This command will hide the wireframe bounds that reveal the collision sphere and collision boundbox associated with the object. These bounds are originally revealed by using the SHOW OBJECT BOUNDS command.

SYNTAX

```
HIDE OBJECT BOUNDS Object Number
```

SET OBJECT
This command sets the internal properties of a specified 3D object number. When the wireframe flag is set to 0, the object only shows its wireframe form. When the transparency flag is set to 0, all parts of the object colored black are not drawn to the screen. When the cull flag is set to 0, the object will draw polygons normally hidden due to the direction the polygon faces. The Filter Value sets the texture filtering, which controls the smoothing effect of the texture as it is mapped to the object. A Filter value of zero does no mipmapping, a value of one uses no smoothing and a value of two uses Linear Filtering. The Light Flag activates and deactivates the objects sensitivity to any lights in the scene. The Fog Flag activates and deactivates the objects sensitivity to fog in the scene. The Ambient Flag activates and deactivates the objects sensitivity to ambient light

in the scene. The object number and flag values should be specified using integer values. If the object uses diffuse based colour, deactivating the light source will also deactivate the ability of the object to show its colour as it uses the light to calculate the diffuse result.

SYNTAX

```
SET OBJECT Object Number, Wire, Transparent, Cull
SET OBJECT Object Number, Wire, Transparent, Cull, Filter
SET OBJECT Object Number, Wire, Transparent, Cull, Filter, Light
SET OBJECT Object Number, Wire, Transparent, Cull, Filter, Light, Fog
SET OBJECT Object Number, Wire, Transparent, Cull, Filter, Light, Fog, Ambient
```

SET OBJECT WIREFRAME
This command will set the wireframe state of the specified object. Setting the wireframe to zero will ensure the object is solid. A value of one will cause the object to draw in wireframe form.

SYNTAX

```
SET OBJECT WIREFRAME Object Number, Flag
```

SET OBJECT TRANSPARENCY
This command will set the transparency state of the specified object. Setting the transparency to one will ensure the object does not draw the transparent colour during final rendering. The transparent colour is determined by the SET IMAGE COLORKEY command.

SYNTAX

```
SET OBJECT TRANSPARENCY Object Number, Flag
```

SET OBJECT CULL
This command will set the cull state of the specified object. Setting the cull to one will ensure the object is culled so that away-facing polygons are not drawn.

SYNTAX

```
SET OBJECT CULL Object Number, Flag
```

SET OBJECT FILTER
This command will set the texture filtering mode of the specified object. The Filter value sets the texture filtering, which controls the smoothing effect of the texture as it is mapped to the object. A Filter value of zero does no mipmapping, a value of one uses no smoothing and a value of two uses Linear Filtering.

SYNTAX

```
SET OBJECT FILTER Object Number, Flag
```

SET OBJECT LIGHT
This command will set the light state of the specified object. Setting the light to one will ensure the object is affected by any lights in the 3D scene. If the object uses diffuse based colour, deactivating the light source will also deactivate the ability of the object to show its colour as it uses the light to calculate the diffuse result.

SYNTAX

```
SET OBJECT LIGHT Object Number, Flag
```

SET OBJECT FOG
This command will set the fog state of the specified object. Setting the fog to one will ensure the object is affected by the global fog level.

SYNTAX

```
SET OBJECT FOG Object Number, Flag
```

SET OBJECT AMBIENT

This command will set the ambient state of the specified object. Setting the ambience to one will ensure the object is affected by the global ambient light level.

SYNTAX

```
SET OBJECT AMBIENT Object Number, Flag
```

SET OBJECT ROTATION XYZ
This command will set the order of rotation to the default behaviour. The angles of rotation will transform the object first around the X axis, then the Y axis and finally the Z axis. The parameter should be specified using an integer value.

SYNTAX

```
SET OBJECT ROTATION XYZ Object Number
```

SET OBJECT ROTATION ZYX
This command will reverse the order of rotation for the specified object. The angles of rotation will transform the object first around the Z axis, then the Y axis and finally the X axis. The parameter should be specified using an integer value.

SYNTAX

```
SET OBJECT ROTATION ZYX Object Number
```

SET OBJECT TO CAMERA ORIENTATION
This command will set the specified 3D object to point in the same direction as the camera. The object number must be specified using an integer value.

SYNTAX

```
SET OBJECT TO CAMERA ORIENTATION Object Number
```

SET OBJECT TO OBJECT ORIENTATION
This command will set the specified 3D object to point in the same direction as another 3D object. The object numbers must be specified using integer values.

SYNTAX

```
SET OBJECT TO OBJECT ORIENTATION Object Number, Second Object
```

DISABLE OBJECT ZDEPTH
This command will make the specified object ignore the Zdepth data and render to the screen over standard polygons. This command is useful for ensuring your FPS Gun will not poke through walls when you get too close.

SYNTAX

```
DISABLE OBJECT ZDEPTH Object Number
```

ENABLE OBJECT ZDEPTH
This command will return the specified object to normal Zdepth operations.

SYNTAX

```
ENABLE OBJECT ZDEPTH Object Number
```

GLUE OBJECT TO LIMB
This command will attach the specified 3D object to a limb of another 3D object. By attaching an object to the limb of another, the objects position, rotation and scale are entirely controlled by the limb. This technique can be used to allow a robot arm to easily grab and lift an item, or allow your hero character to hold and wear a variety of items. The parameters should be specified using integer values.

SYNTAX

```
GLUE OBJECT TO LIMB Object Number, Second Object, Limb Number
```

UNGLUE OBJECT
This command will detach the specified 3D object from any limb. The parameter should be specified using an integer value.

SYNTAX

```
UNGLUE OBJECT Object Number
```

FIX OBJECT PIVOT
This command will fix the current angles of the specified 3D object as the new absolute rotation of the model. It is often required to load, rotate and fix models to face a particular direction before using them. The object number should be specified using an integer value.

SYNTAX

```
FIX OBJECT PIVOT Object Number
```

SET OBJECT DIFFUSE
This command will alter the material diffuse component of the object. The diffuse component makes up one of the factors that controls the final colour of the object, specifically the amount of light it receives from all available light sources.

SYNTAX

```
SET OBJECT DIFFUSE Object Number, Color Value
```

SET OBJECT AMBIENCE
This command will alter the material ambience component of the object. The ambience component makes up one of the factors that controls the final colour of the object, specifically the amount of global ambient light it receives. If the global ambient was red, and this component was set to white, the object would receive 'full' ambient light and drown out any lighting otherwise applied to the object. The object would be completely red. This command should not be confused with SET OBJECT AMBIENT which controls the flag which states whether the object receives any ambient light whatsoever.

SYNTAX

```
SET OBJECT AMBIENCE Object Number, Color Value
```

SET OBJECT SPECULAR
This command will alter the material specular component of the object. The specular component makes up one of the factors that controls the final colour of the object, specifically the amount of light to be reflected from the surface of the object. The strength of the specular reflection is controlled with the sister command SET OBJECT SPECULAR POWER.

SYNTAX

```
SET OBJECT SPECULAR Object Number, Color Value
```

SET OBJECT EMISSIVE
This command will alter the material emissive component of the object. The emissive component makes up one of the factors that controls the final colour of the object, specifically the colour the object emits irrespective of any light or ambience. Setting this value to a high colour will make it bright under its own inner colour.

SYNTAX

```
SET OBJECT EMISSIVE Object Number, Color Value
```

SET OBJECT SPECULAR POWER
This command will alter the material specular power component of the object. The specular power component makes up one of the factors that controls the final colour of the object, specifically the strength of light reflected from the surface of the object. The amount of the specular reflection is controlled with the sister command SET OBJECT SPECULAR.

SYNTAX

```
SET OBJECT SPECULAR POWER Object Number, Power
```

SET LIGHT MAPPING ON
This command will set the light map for the specified object. Light map textures are combined with the main texture to create a multitextured object.

SYNTAX

```
SET LIGHT MAPPING ON Object Number, Image Number
```

SET DETAIL MAPPING ON
This command will set the detail map for the specified object. Detail map textures are combined with the main texture to create a multitextured object.

SYNTAX
```
SET DETAIL MAPPING ON Object Number, Image Number
```

SET BLEND MAPPING ON
This command will set the blending map for the specified object. Blended textures are combined with the main texture to create a multitextured object. The Blend value is the mode used to create the blend effect.

SYNTAX
```
SET BLEND MAPPING ON Object Number, Image Number, Blend Mode
```

SET SPHERE MAPPING ON
This command will apply a sphere map to the specified object. A sphere map will give the impression of a reflective surface over the object. The image used for the sphere map is specially prepared to be used in concert with this command. The image is a fisheye spherical view of an entire scene placed in the center of the texture plate.

SYNTAX
```
SET SPHERE MAPPING ON Object Number, Image Number
```

SET CUBE MAPPING ON
This command will apply a cube map to the specified object. A cube map will give the impression of a reflective surface over the object. The images used specify a texture on each side of the object to be affected, which combine to create a 3D texture from which the reflective parts can be mapped to the object. Cube maps take up a lot of video memory so use this feature with care.

SYNTAX
```
SET CUBE MAPPING ON Object Number, Face1, Face2, Face3, Face4, Face5, Face6
```

SET BUMP MAPPING ON
This command will set the bump map for the specified object. Bumpmapped textures are combined with the main texture to create an object with a visually uneven surface. Your video card must have the ability to create this hardware effect.

SYNTAX
```
SET BUMP MAPPING ON Object Number, Image Number
```

SET CARTOON SHADING ON
This command will set the rendering method of the specified object. Otherwise known as toon shading, this command will apply a carton style render to the object using a shading image and an edge image. The shading image and edge image require layout for the toon shading to work correctly. Your video card must have the ability to create this hardware effect.

SYNTAX
```
SET CARTOON SHADING ON Object Number, Shade Image, Edge Image
```

SET RAINBOW SHADING ON
This command will set the specified object to use rainbow rendering. By specifying a rainbow image, you can create a very colourful rainbow effect for your object. Your video card must have the ability to create this hardware effect.

SYNTAX
```
SET RAINBOW SHADING ON Object Number, Rainbow Image
```

SET SHADOW SHADING ON
This command will set the specified object to cast a hardware shadow. This command requires hardware specifically designed for this feature, as it's a very intensive calculation. Use this command sparingly for best performance.

SYNTAX

```
SET SHADOW SHADING ON Object Number
```

SET REFLECTION SHADING ON
This command will set the specified object to use reflection. Use this command when you want to give your object a reflective quality, though not all 3D hardware will support Vertex Shaders which are required to produce this effect.

SYNTAX

```
SET REFLECTION SHADING ON Object Number
```

SET ALPHA MAPPING ON
This command will set the true alpha value of an object to a percentage value from 0 to 100. Zero represents an alpha state that makes the object completely invisible. An alpha percentage of 100 will render the object fully visible. A value between these limits will create a true transparency effect when the object is transparent, as set by the SET OBJECT TRANSPARENCY command.

SYNTAX

```
SET ALPHA MAPPING ON Object Number, Percentage
```

SET EFFECT ON
This command will load and apply an FX file to an object. If the object does not exist onto which an effect is set, a default model is created either specified by the FX file or if none is specified, an internal pyramid model. The FX filename must point to an FX file that is compatible with the DX9 effect framework and typically uses the extension '.fx'. If the Texture Flag is set to zero, the effect will use the textures already mapped to the model, and a value of one will discard the current textures and load the textures specified in the FX file. Typically, effects rely on the specified model and textures in the FX file to function properly.

SYNTAX

```
SET EFFECT ON Object Number, FX Filename, Texture Flag
```

SET SHADING OFF
This command will deactivate any shading currently being applied to the specified object. Shading activated by such commands as SET SHADOW SHADING ON will be switched off restoring the object to a regular object.

SYNTAX

```
SET SHADING OFF Object Number
```

LOAD EFFECT
This command will load an FX file.

SYNTAX

```
LOAD EFFECT Filename, Effect Number, Texture Flag
```

DELETE EFFECT
This command will delete an FX effect. Any effects that are using this effect will cease to render correctly if the FX is deleted while it is being used by the application.

SYNTAX

```
DELETE EFFECT Effect Number
```

SET OBJECT EFFECT
This command will apply a previously loaded FX effect onto the specified object. Using this command instead of SET EFFECT ON allows many objects to share a single FX system and increase the performance of your application.

SYNTAX

```
SET OBJECT EFFECT Object Number, Effect Number
```

SET LIMB EFFECT

This command will apply a previously loaded FX effect onto the specified limb of an object. You can use this command to apply an FX effect to a single part of a model. You can apply different effects to different parts of the same model to create some stunning results.

SYNTAX

SET LIMB EFFECT Object Number, Limb Number, Effect Number

PERFORM CHECKLIST FOR EFFECT VALUES
This command will create a checklist of all the effect constants used by the loaded FX file. The checklist strings will contain the names of the constants, which can then be individually set with the SET EFFECT CONSTANT command.

SYNTAX

PERFORM CHECKLIST FOR EFFECT VALUES Effect Number

PERFORM CHECKLIST FOR EFFECT ERRORS
This command will create a checklist an error report upon failure to load an FX file. The checklist strings will contain a single line of the report, for however large the report is.

SYNTAX

PERFORM CHECKLIST FOR EFFECT ERRORS Effect Number

SET EFFECT CONSTANT BOOLEAN
This command will set the value of an FX effect boolean constant. An FX effect constant is one of many internal variables of the effect which control the effect in real-time based on what the FX does. Some constants are automatically provided to the FX system effect such as world and camera positions, time and other critical data. Often, FX files will contain extra constants to create a variety of results and these constants can be altered in real-time by this command. Use the PERFORM CHECKLIST FOR EFFECT VALUES to get the names of all the constants you can alter.

SYNTAX

SET EFFECT CONSTANT BOOLEAN Effect Number, Constant String, Constant Value

SET EFFECT CONSTANT INTEGER
This command will set the value of an FX effect integer constant. An FX effect constant is one of many internal variables of the effect which control the effect in real-time based on what the FX does. Some constants are automatically provided to the FX system effect such as world and camera positions, time and other critical data. Often, FX files will contain extra constants to create a variety of results and these constants can be altered in real-time by this command. Use the PERFORM CHECKLIST FOR EFFECT VALUES to get the names of all the constants you can alter.

SYNTAX

SET EFFECT CONSTANT INTEGER Effect Number, Constant String, Constant Value

SET EFFECT CONSTANT FLOAT
This command will set the value of an FX effect float constant. An FX effect constant is one of many internal variables of the effect which control the effect in real-time based on what the FX does. Some constants are automatically provided to the FX system effect such as world and camera positions, time and other critical data. Often, FX files will contain extra constants to create a variety of results and these constants can be altered in real-time by this command. Use the PERFORM CHECKLIST FOR EFFECT VALUES to get the names of all the constants you can alter.

SYNTAX

SET EFFECT CONSTANT FLOAT Effect Number, Constant String, Constant Value

SET EFFECT CONSTANT VECTOR
This command will set the value of an FX effect vector constant. An FX effect constant is one of many internal variables of the effect which control the effect in real-time based on what the FX does. Some constants are automatically provided to the FX system effect such as world and camera positions, time and other critical data. Often, FX files will contain extra constants to create a variety of results and these constants can be altered in real-time by this command. Use the PERFORM CHECKLIST FOR EFFECT VALUES to get the names of all the constants you can alter.

SYNTAX

SET EFFECT CONSTANT VECTOR Effect Number, Constant String, Vector Number

SET EFFECT CONSTANT MATRIX
This command will set the value of an FX effect matrix constant. An FX effect constant is one of many internal variables of the effect which control the effect in real-time based on what the FX does. Some constants are automatically provided to the FX system effect such as world and camera positions, time and other critical data. Often, FX files will contain extra constants to create a variety of results and these constants can be altered in real-time by this command. Use the PERFORM CHECKLIST FOR EFFECT VALUES to get the names of all the constants you can alter.

SYNTAX

```
SET EFFECT CONSTANT MATRIX Effect Number, Constant String, Matrix Number
```

SET EFFECT TECHNIQUE
This command will set the technique of the specified FX effect. The technique is specified by name, and the name can be found heading the technique within the FX file itself. An FX effect can have multiple techniques within the file.

SYNTAX

```
SET EFFECT TECHNIQUE Effect Number, Technique Name
```

SET EFFECT TRANSPOSE
This command will set the transpose method of the FX effect. Low and high level shaders perform 3D maths using the matrix data passed in by the application. Some of these shaders require that the matrix data is tranposed, where others require untransposed matrix data. Setting the transpose flag to one will tranpose any matrix data passed to the shader.

SYNTAX

```
SET EFFECT TRANSPOSE Effect Number, Transpose Flag
```

COLLISION COMMANDS

MAKE OBJECT COLLISION BOX
This command will make an object use box collision. If a value of zero is specified for the flag value, the box will be deemed to be a non-rotating box and able to generate sliding collision data using the GET OBJECT COLLISION X, Y and Z expressions. If the flag value is one, the box is deemed to be a rotated-box and will provide more accurate collision feedback as the object rotates.

SYNTAX

```
MAKE OBJECT COLLISION BOX Object Number, X1, Y1, Z1, X2, Y2, Z2, Collision Flag
```

DELETE OBJECT COLLISION BOX
This command will make an object use normal collision.

SYNTAX

```
DELETE OBJECT COLLISION BOX Object Number
```

SET OBJECT COLLISION ON
This command will set the specified 3D object to detect for and be detected for any collisions that occur. The parameter should be specified using an integer value.

SYNTAX

```
SET OBJECT COLLISION ON Object Number
```

SET OBJECT COLLISION OFF
This command will set the specified 3D object to ignore and be ignored should any collision occur. The parameter should be specified using an integer value.

SYNTAX

```
SET OBJECT COLLISION OFF Object Number
```

SET OBJECT COLLISION TO BOXES

This command will set the specified 3D object to use a box area for collision detection. An invisible collision box will be used for every limb in the object. The parameter should be specified using an integer value.

SYNTAX

```
SET OBJECT COLLISION TO BOXES Object Number
```

SET OBJECT COLLISION TO POLYGONS
This command will set the specified 3D object to use polygon checking for collision detection. Polygon detection is much slower that box and sphere detection, but allows you to detect perfect collision against an object that has a complex polygon structure. The parameter should be specified using an integer value. When a polygon test is carried out, the primary object is always treated as a polygon object and the secondary object is always a sphere representative of the size of the second object. This method is significantly faster that a test against two polygon structures.

SYNTAX

```
SET OBJECT COLLISION TO POLYGONS Object Number
```

SET OBJECT COLLISION TO SPHERES
This command will set the specified 3D object to use a spherical area for collision detection. An invisible collision sphere will be used for every limb in the object. The parameter should be specified using an integer value.

SYNTAX

```
SET OBJECT COLLISION TO SPHERES Object Number
```

SET GLOBAL COLLISION ON
This command will activate the detection of collisions with any two 3D object, irrespective of their individual collision settings.

SYNTAX

```
SET GLOBAL COLLISION ON
```

SET GLOBAL COLLISION OFF
This command will deactivate the influence of global collision. No 3D objects will be able to detect collisions with each other, unless the object has individual collision detection active.

SYNTAX

```
SET GLOBAL COLLISION OFF
```

SET OBJECT RADIUS
This command will set the collision radius of the specified object. You can use this command to remove the outer most parts of your object from the collision system where collision at such extents are undesirable.

SYNTAX

```
SET OBJECT RADIUS Object Number, Radius
```

AUTOMATIC OBJECT COLLISION
This command will set the specified object to automatic collision. Automatic collision takes over the task of adjusting the object when it hits another object in the 3D scene. Objects with automatic collision use a collision sphere of the specified radius, ignoring its previous collision shape. If the Response value is set to one, the new position backtracks to the last collision free position when a hit occurs.

SYNTAX

```
AUTOMATIC OBJECT COLLISION Object Number, Radius, Response
```

AUTOMATIC CAMERA COLLISION
This command will set the specified camera to automatic collision. Automatic collision takes over the task of adjusting the camera when it hits an object in the 3D scene. Cameras with automatic collision use a collision sphere of the specified radius. If the Response value is set to one, the new position backtracks to the last collision free position when a hit occurs.

SYNTAX

```
AUTOMATIC CAMERA COLLISION Camera Number, Radius, Response
```

MAKE STATIC COLLISION BOX
This command will create invisible collision zones within the static world data. Once defined, you can use the GET STATIC COLLISION HIT command to detect whether another dynamically specified box area collides with this one. This command is ideally used to define solid areas within your 3D world.

SYNTAX

```
MAKE STATIC COLLISION BOX X1, Y1, Z1, X2, Y2, Z2
```

LIMB COMMANDS

PERFORM CHECKLIST FOR OBJECT LIMBS
This command will make a list of all limbs contained in a 3D object. Objects that have been loaded may contain hundreds of limbs, and are identified by number. All limbs include an internal limb description that often indicates which part of the overall 3D object it belongs to. You can access the limb description using the string item of the checklist when you have performed the check. Use the CHECKLIST commands in the SYSTEM command set to read the checklist.

SYNTAX

```
PERFORM CHECKLIST FOR OBJECT LIMBS Object Number
```

OFFSET LIMB
This command will change the relative position of the specified limb within the 3D object. The position of the limb is always offset from the main coordinates of the 3D object and from any parent limbs. Specifying a limb number of zero provides access to the objects own root data, and should not normally be used in this way. The object and limb parameters should be specified using integer values. The offset parameters should be specified using real numbers.

SYNTAX

```
OFFSET LIMB Object Number, Limb Number, X, Y, Z
```

SCALE LIMB
This command will change the scale of the specified limb within the 3D object by affecting the percentage scale value of all three dimensions. Specifying a limb number of zero provides access to the objects own root data, and should not normally be used in this way. The parameters should be specified using integer values.

SYNTAX

```
SCALE LIMB Object Number, Limb Number, XSize, YSize, ZSize
```

ROTATE LIMB
This command will change the rotation of the specified limb within the 3D object. Specifying a limb number of zero provides access to the objects own root data, and should not normally be used in this way. The object and limb parameters should be specified using integer values. The offset parameters should be specified using real numbers.

SYNTAX

```
ROTATE LIMB Object Number, Limb Number, XAngle, YAngle, ZAngle
```

SHOW LIMB
This command will show the specified limb within the 3D object previously hidden. The parameters should be specified using integer values.

SYNTAX

```
SHOW LIMB Object Number, Limb Number
```

HIDE LIMB
This command will hide the specified limb within the 3D object.

SYNTAX

```
HIDE LIMB Object Number, Limb Number
```

TEXTURE LIMB
This command will apply an existing image to the limb of a 3D object as a texture. You must create an image first using the GET IMAGE command before attempting to texture part of the 3D object. The parameters should be specified using integer values. An additional texture stage index can be provided to the command to specify multi-textures directly, and is required when using shaders that take pixel data from secondary textures.

SYNTAX

TEXTURE LIMB Object Number, Limb Number, Image Number

COLOR LIMB
This command will color the specified limb of a 3D object using an RGB colour value. The parameters must be integer values. The RGB color value can be generated by using the RGB command. Some objects loaded from a model file may not be colored if they do not already contain diffuse color data.

SYNTAX

COLOR LIMB Object Number, Limb Number, Color Value

SCALE LIMB TEXTURE
This command will scale the UV data of the specified limb of the 3D object. The UV data controls how a texture is mapped onto your object. By scaling the UV data, you can effectively stretch or tile the texture over your object. The U value controls the horizontal spread of the data. The V value controls the vertical spread of the data. A U or V value of 1 means no scale change. A value of 0.5 will scale the texture by half. A value of 2.0 will double the scale of the texture. The scale effect is permanent.

SYNTAX

SCALE LIMB TEXTURE Object Number, Limb Number, UScale, VScale

SCROLL LIMB TEXTURE
This command will scroll the UV data of the specified limb of the 3D object. The UV data controls how a texture is mapped onto your object. By scrolling the UV data, you can effectively scroll the texture over your limb. The U value controls the horizontal shift of the data. The V value controls the vertical shift of the data. The scroll effect is permanent.

SYNTAX

SCROLL LIMB TEXTURE Object Number, Limb Number, X, Y

ADD LIMB
This command will create a new limb from a specified mesh and add it to an existing 3D object. Limbs can only be added sequentially, so you must ensure you specify a new limb number that immediately follows an existing limb. The parameters should be specified using integer values. When a limb is added to a 3D object, it will not have a place in the object hierarchy. You can position the limb in the object hierarchy using the LINK LIMB command. Do not confuse LINK LIMB with OFFSET LIMB which sets the actual 3D position of the limb within the object.

SYNTAX

ADD LIMB Object Number, Limb Number, Mesh Number

LINK LIMB
This command will link a newly created limb to a limb within an existing 3D object. When a limb is connected to another, it becomes a child limb that will be affected by the position, rotation and scale of its parent limbs. The parameters should be specified using integer values.

SYNTAX

LINK LIMB Object Number, Limb Parent, Limb Child

REMOVE LIMB
This command will remove an existing limb from an object. Destroying a limb will also remove every limb associated as a child node from the removed limb.

SYNTAX

REMOVE LIMB Object Number, Limb Number

SET LIMB SMOOTHING
This command will smooth the sharp edges of the mesh within a limb by adjusting the normals data. A percentage value of zero will perform no smoothing and create a facet surface for the limb. A percentage value of 100 will perform full smoothing, averaging all normals that share a vertex position and create a smoothing effect elimiating all edges. A value between these two limits will determine the degree beyond which an edge will be smoothed or left sharp.

SYNTAX

```
SET LIMB SMOOTHING Object Number, Limb Number, Percentage
```

MESH COMMANDS

LOAD MESH
This command will load a single X file into the specified mesh number. A mesh is a wireframe description of a 3D shape. You must use a filename that points to a file that stores 3D mesh data in the X file format. The mesh number should be specified using an integer value.

SYNTAX

```
LOAD MESH Filename, Mesh Number
```

SAVE MESH
This command will save the specified mesh to a file in the XFile text format. You can use this command to extract and store meshes previously existing as part of a more complex object, or to create a large mesh based world and save it out for later loading as a 3D game level.

SYNTAX

```
SAVE MESH Filename, Mesh Number
```

DELETE MESH
This command will delete the specified mesh previously loaded. Deleting unused meshes improves system performance. After you have used a mesh to create an object or limb, you are free to delete it. The parameter should be specified using an integer value.

SYNTAX

```
DELETE MESH Mesh Number
```

CHANGE MESH
This command will change the mesh of an object limb. You can use this command to animate an object that requires a sequence of fixed static meshes. The parameters must be integer values.

SYNTAX

```
CHANGE MESH Object Number, Limb Number, Mesh Number
```

MAKE MESH FROM OBJECT
This command will make a single mesh using the entire mesh data of an object. A mesh is a wireframe description of a 3D shape. The mesh and object numbers should be specified using integer values.

SYNTAX

```
MAKE MESH FROM OBJECT Mesh Number, Object Number
```

SHADER COMMANDS

CREATE VERTEX SHADER FROM FILE
This command will create a vertex shader from a specified text file. Use this command in concert with other vertex shader commands to create a special effect on a 3D object. Vertex shaders are powerful programmable effects, and not all 3D devices support them. Make sure that your shader text file includes the minimum amount of information with your code in order that assembly can take place. Ensure you are using the correct versioning label and that the input vertex data format is specified. An example of this syntax might be 'vs.1.0 dcl_position v0 dcl_normal v3 dcl_color v6 dcl_texcoord0 v7'.

SYNTAX

CREATE VERTEX SHADER FROM FILE VertexShader Number, Shader Filename

DELETE VERTEX SHADER
This command will delete the specified vertex shader if one exists. Delete vertex shaders when you have no further use for them to save a small amount of memory.

SYNTAX

DELETE VERTEX SHADER VertexShader Number

SET VERTEX SHADER ON
This command will assign a vertex shader to an object. If the vertex shader is valid, the shader effect will be applied immediately to the object.

SYNTAX

SET VERTEX SHADER ON Object Number, VertexShader Number

SET VERTEX SHADER OFF
This command will remove the influence of the vertex shader from the object. The vertex shader is not deleted, or deactivated, only the specified object will be affected.

SYNTAX

SET VERTEX SHADER OFF Object Number

SET VERTEX SHADER VECTOR
This command will select the vector to be used by the specified vertex shader.

SYNTAX

SET VERTEX SHADER VECTOR VertexShader Number, Constant, Vector4, ConstantCount

SET VERTEX SHADER MATRIX
This command will select the matrix to be used by the specified vertex shader.

SYNTAX

SET VERTEX SHADER MATRIX VertexShader Number, Constant, Matrix4, ConstantCount

SET VERTEX SHADER STREAM
This command will set a vertex shader stream for the specified vertex shader. The stream index must start from one, being the first stream into the shader. The Data is specified using one of the following values, which relate to the FVF Format code you would use to convert your object prior to using a shader:VSDE_POSITION=FVF_XYZ=0, VSDE_BLENDWEIGHT=FVF_XYZRHW=1, VSDE_BLENDINDICES=FVF_XYZB1=2, VSDE_NORMAL=FVF_NORMAL=3, VSDE_PSIZE=FVF_PSIZE=4, VSDE_DIFFUSE=FVF_DIFFUSE=5, VSDE_SPECULAR=FVF_SPECULAR=6, VSDE_TEXCOORD0=FVF_TEX0=7, VSDE_TEXCOORD1=FVF_TEX0=8, VSDE_TEXCOORD2=FVF_TEX0=9, VSDE_TEXCOORD3=FVF_TEX0=10, VSDE_TEXCOORD4=FVF_TEX0=11, VSDE_TEXCOORD5=FVF_TEX0=12, VSDE_TEXCOORD6=FVF_TEX0=13, VSDE_TEXCOORD7=FVF_TEX0=14. The Datatype is specified using one of the following values:

SYNTAX

SET VERTEX SHADER STREAM VertexShader Number, Position, Data, Datatype

SET VERTEX SHADER STREAMCOUNT
This command will set the vertex shader stream count for the specified vertex shader.

SYNTAX

SET VERTEX SHADER STREAMCOUNT VertexShader Number, Count

CONVERT OBJECT FVF
This command will convert the specified object to use a new FVF Format. FVF stands for Flexible Vertex Format and represents how much data each vertex of your objects meshes contain. Use this command to prepare an object for use with

a vertex shader effect. The FVF Format is specified using bits. Each bit represents an element that will be created within the vertex data. The FVF bits are specified within a DWORD and are as follows: FVF_XYZ=0x002, FVF_XYZRHW=0x004, FVF_XYZB1=0x006, FVF_XYZB2=0x008, FVF_XYZB3=0x00a, FVF_XYZB4=0x00c, FVF_XYZB5=0x00e, FVF_NORMAL=0x010, FVF_PSIZE=0x020, FVF_DIFFUSE=0x040, FVF_SPECULAR=0x080, FVF_TEX0=0x000, FVF_TEX1=0x100, FVF_TEX2=0x200, FVF_TEX3=0x300, FVF_TEX4=0x400, FVF_TEX5=0x500, FVF_TEX6=0x600, FVF_TEX7=0x700, FVF_TEX8=0x800.

SYNTAX

CONVERT OBJECT FVF Object Number, FVF Format

CREATE PIXEL SHADER FROM FILE
This command will create a pixel shader from a specified text file. Use this command in concert with other pixel shader commands to create a special effect on a 3D object. Pixel shaders are powerful programmable effects, and not all 3D devices support them.

SYNTAX

CREATE PIXEL SHADER FROM FILE PixelShader Number, Shader Filename

DELETE PIXEL SHADER
This command will delete the specified pixel shader if one exists. Delete pixel shaders when you have no further use for them to save a small amount of memory.

SYNTAX

DELETE PIXEL SHADER PixelShader Number

SET PIXEL SHADER ON
This command will assign a pixel shader to an object. If the pixel shader is valid, the shader effect will be applied immediately to the object.

SYNTAX

SET PIXEL SHADER ON Object Number, PixelShader Number

SET PIXEL SHADER OFF
This command will remove the influence of the pixel shader from the object. The pixel shader is not deleted, or deactivated, only the specified object will be affected.

SYNTAX

SET PIXEL SHADER OFF Object Number

SET PIXEL SHADER TEXTURE
This command will select the image to be used by the specified pixel shader.

SYNTAX

SET PIXEL SHADER TEXTURE PixelShader Number, Slot Number, Image Number

FLUSH VIDEO MEMORY
This command will flush all textures and other resources from active video memory. You should use this command when you are dramatically changing the contents of your 3D scene such as changing game levels or entering a new environment. This technique will improve performance during the initial renders of a new scene.

SYNTAX

FLUSH VIDEO MEMORY

COMPILE CSG
This command will convert a specially designed X file into a CSG processed geometry set and save it out as an output X file. The input X file has to consist of a series of solid meshes with no complex frame hierarchy. The Constructive Solid Geometry process will split these meshes to remove all hidden surfaces. This can save polygons in large scenes, but may also increase polygon counts if the geometry is complex. A typical example of a good input X file is a sequence of solid boxes placed side by side to form a wall. CSG would remove the polygons where the boxes touch reducing the polygon count and

improving polygon usage.

SYNTAX
```
COMPILE CSG Input X File, Output X File
COMPILE CSG Input X File, Output X File, Epsilon
```

OBJECT EXPRESSIONS

OBJECT EXIST
This command will return a one if the specified 3D object exists, otherwise zero will be returned. The parameter should be specified using an integer value.

SYNTAX
```
Return Integer=OBJECT EXIST(Object Number)
```

OBJECT VISIBLE
This command will return a one if the specified 3D object is visible, otherwise zero will be returned. The parameter should be specified using an integer value.

SYNTAX
```
Return Integer=OBJECT VISIBLE(Object Number)
```

OBJECT PLAYING
This command will return a one if the specified 3D object is playing its animation, otherwise zero will be returned. The parameter should be specified using an integer value.

SYNTAX
```
Return Integer=OBJECT PLAYING(Object Number)
```

OBJECT LOOPING
This command will return a one if the specified 3D object is looping its animation, otherwise zero will be returned. The parameter should be specified using an integer value.

SYNTAX
```
Return Integer=OBJECT LOOPING(Object Number)
```

OBJECT POSITION X
This command will return a real value X position of the specified 3D object in 3D space. The parameter should be specified using an integer value.

SYNTAX
```
Return Float=OBJECT POSITION X(Object Number)
```

OBJECT POSITION Y
This command will return a real value Y position of the specified 3D object in 3D space. The parameter should be specified using an integer value.

SYNTAX
```
Return Float=OBJECT POSITION Y(Object Number)
```

OBJECT POSITION Z
This command will return a real value Z position of the specified 3D object in 3D space. The parameter should be specified using an integer value.

SYNTAX
```
Return Float=OBJECT POSITION Z(Object Number)
```

OBJECT SIZE
This command will return a real number representing the full unit size of the specified 3D object. The unit size can be used to determine whether or not to scale the object for better visibility. Extremely small and extremely large objects will both suffer visual clipping when viewed by the camera. As a rule, your objects should have a unit size of between 50 and 3000. The finite visibility of the camera has a range of 5000 units, and objects of a distance greater than this will be clipped. Objects that are so close to the camera that they pass behind the camera will also be clipped. The parameter should be specified using an integer value.

SYNTAX

Return Float=OBJECT SIZE(Object Number)

OBJECT SIZE X
This command will return the full width of the object.

SYNTAX

Return Float=OBJECT SIZE X(Object Number)

OBJECT SIZE Y
This command will return the full height of the object.

SYNTAX

Return Float=OBJECT SIZE Y(Object Number)

OBJECT SIZE Z
This command will return the full depth of the object.

SYNTAX

Return Float=OBJECT SIZE Z(Object Number)

OBJECT ANGLE X
This command will return a real value X angle of the specified 3D object. The parameter should be specified using an integer value.

SYNTAX

Return Float=OBJECT ANGLE X(Object Number)

OBJECT ANGLE Y
This command will return a real value Y angle of the specified 3D object. The parameter should be specified using an integer value.

SYNTAX

Return Float=OBJECT ANGLE Y(Object Number)

OBJECT ANGLE Z
This command will return a real value Z angle of the specified 3D object. The parameter should be specified using an integer value.

SYNTAX

Return Float=OBJECT ANGLE Z(Object Number)

OBJECT FRAME
This command will return a real value of the current animation frame of the specified 3D object. Animation frames are counted by keyframe represented as whole numbers, but the animation can be anywhere between two keyframes which is why the frame is returned using a real value. The parameter should be specified using an integer value.

SYNTAX

Return Integer=OBJECT FRAME(Object Number)

OBJECT SPEED
This command will return an integer value of the current animation speed of the specified 3D object. The parameter should be specified using an integer value.

SYNTAX
```
Return Integer=OBJECT SPEED(Object Number)
```

OBJECT INTERPOLATION
This command will return an integer of the current animation interpolation percentage of the specified 3D object. The parameter should be specified using an integer value.

SYNTAX
```
Return Integer=OBJECT INTERPOLATION(Object Number)
```

TOTAL OBJECT FRAMES
This command will return an integer value of the last known animation frame number of the specified 3D object. The parameter should be specified using an integer value.

SYNTAX
```
Return Integer=TOTAL OBJECT FRAMES(Object Number)
```

COLLISION EXPRESSIONS

OBJECT HIT
This command will return a one if the two specified 3D objects hit each other. If the second object number is set to zero, the number of the object hit by the first object will be returned as an integer value. The parameters should be specified using integer values.

SYNTAX
```
Return Integer=OBJECT HIT(ObjectA Number, ObjectB Number)
```

OBJECT COLLISION
This command will return a one if the two specified 3D objects are overlapping. If the second object number is set to zero, the number of the object overlapping with the first object will be returned as an integer value. The parameters should be specified using integer values.

SYNTAX
```
Return Integer=OBJECT COLLISION(ObjectA Number, ObjectB Number)
```

GET OBJECT COLLISION X
This command will return sliding data for X if the rotated-box flag is zero. The rotated-box flag is set when you used the command MAKE OBJECT COLLISION BOX. You must have used the make command in order to generate a return value here.

SYNTAX
```
Return Float=GET OBJECT COLLISION X()
```

GET OBJECT COLLISION Y
This command will return sliding data for Y if the rotated-box flag is zero. The rotated-box flag is set when you used the command MAKE OBJECT COLLISION BOX. You must have used the make command in order to generate a return value here.

SYNTAX
```
Return Float=GET OBJECT COLLISION Y()
```

GET OBJECT COLLISION Z
This command will return sliding data for Z if the rotated-box flag is zero. The rotated-box flag is set when you used the command MAKE OBJECT COLLISION BOX. You must have used the make command in order to generate a return value here.

SYNTAX

Return Float=GET OBJECT COLLISION Z()

OBJECT COLLISION RADIUS
This command will return the radius of the collision sphere of the specified object. The radius is returned as a real value, and the object is specified using an integer value.

SYNTAX

Return Float=OBJECT COLLISION RADIUS(Object Number)

OBJECT COLLISION CENTER X
This command will return the X position of the center of the collision sphere associated with the specified object. The position is returned as a real value, and the object is specified using an integer value.

SYNTAX

Return Float=OBJECT COLLISION CENTER X(Object Number)

OBJECT COLLISION CENTER Y
This command will return the Y position of the center of the collision sphere associated with the specified object. The position is returned as a real value, and the object is specified using an integer value.

SYNTAX

Return Float=OBJECT COLLISION CENTER Y(Object Number)

OBJECT COLLISION CENTER Z
This command will return the Z position of the center of the collision sphere associated with the specified object. The position is returned as a real value, and the object is specified using an integer value.

SYNTAX

Return Float=OBJECT COLLISION CENTER Z(Object Number)

INTERSECT OBJECT
This command will return the distance to the point of intersection between two coordinates, in reference to the specified object. Use this command to project a line from your current position to a destination to determine whether a collision will occur with an object. Ideal for bullet calculations and fast manual polygon collision.

SYNTAX

Return Float=INTERSECT OBJECT(Object Number, X, Y, Z, ToX, ToY, ToZ)

LIMB EXPRESSIONS

LIMB EXIST
This command will return a one if the specified 3D object exists, otherwise zero will be returned.

SYNTAX

Return Integer=LIMB EXIST(Object Number, Limb Number)

LIMB NAME$
This command will return the name string of the specified limb. You can also get the limb name by running a checklist on an object, however this interrogates an individual limb number for its actual name within the object.

SYNTAX

Return String=LIMB NAME$(Object Number, Limb Number)

LIMB VISIBLE
This command will return a one if the specified 3D object exists, otherwise zero will be returned. The parameters should be

specified using integer values.

SYNTAX

Return Integer=LIMB VISIBLE(Object Number, Limb Number)

LIMB OFFSET X

This command will return a real number X offset of the specified limb of the 3D object. Specifying a limb number of zero provides access to the objects own root data, and should not normally be used in this way. The position of the limb is always offset from the positional Coordinates of the 3D object itself.

SYNTAX

Return Float=LIMB OFFSET X(Object Number, Limb Number)

LIMB OFFSET Y

This command will return a real number Y offset of the specified limb of the 3D object. Specifying a limb number of zero provides access to the objects own root data, and should not normally be used in this way. The position of the limb is always offset from the positional coordinates of the 3D object itself.

SYNTAX

Return Float=LIMB OFFSET Y(Object Number, Limb Number)

LIMB OFFSET Z

This command will return a real number Z offset of the specified limb of the 3D object. Specifying a limb number of zero provides access to the objects own root data, and should not normally be used in this way. The position of the limb is always offset from the positional coordinates of the 3D object itself.

SYNTAX

Return Float=LIMB OFFSET Z(Object Number, Limb Number)

LIMB SCALE X

This command will return the X scale value of the limb, normally set to 100 as a percentage indicator. A value of 50 would represent half scale, and a value of 200 would be double the scale.

SYNTAX

Return Float=LIMB SCALE X(Object Number, Limb Number)

LIMB SCALE Y

This command will return the Y scale value of the limb, normally set to 100 as a percentage indicator. A value of 50 would represent half scale, and a value of 200 would be double the scale.

SYNTAX

Return Float=LIMB SCALE Y(Object Number, Limb Number)

LIMB SCALE Z

This command will return the Z scale value of the limb, normally set to 100 as a percentage indicator. A value of 50 would represent half scale, and a value of 200 would be double the scale.

SYNTAX

Return Float=LIMB SCALE Z(Object Number, Limb Number)

LIMB ANGLE X

This command will return the real number X angle of the specified limb of the 3D object. Specifying a limb number of zero provides access to the objects own root data, and should not normally be used in this way. The parameters should be specified using integer values.

SYNTAX

Return Float=LIMB ANGLE X(Object Number, Limb Number)

LIMB ANGLE Y
This command will return the real number Y angle of the specified limb of the 3D object. Specifying a limb number of zero provides access to the objects own root data, and should not normally be used in this way. The parameters should be specified using integer values.

SYNTAX
Return Float=LIMB ANGLE Y(Object Number, Limb Number)

LIMB ANGLE Z
This command will return the real number Z angle of the specified limb of the 3D object. Specifying a limb number of zero provides access to the objects own root data, and should not normally be used in this way. The parameters should be specified using integer values.

SYNTAX
Return Float=LIMB ANGLE Z(Object Number, Limb Number)

LIMB TEXTURE
This command will return the integer image number used to texture the specified limb of the 3D object. Limbs that have been loaded pre-textured contain internal image data and return an image number of zero. The parameters should be specified using integer values.

SYNTAX
Return Integer=LIMB TEXTURE(Object Number, Limb Number)

LIMB TEXTURE NAME
This command will return the internal name of the limb texture.

SYNTAX
Return String=LIMB TEXTURE NAME(Object Number, Limb Number)

LIMB DIRECTION X
This command will return the real number world X angle of the specified limb of the 3D object. Specifying a limb number of zero provides access to the objects own root data, and should not normally be used in this way. The parameters should be specified using integer values.

SYNTAX
Return Float=LIMB DIRECTION X(Object Number, Limb Number)

LIMB DIRECTION Y
This command will return the real number world Y angle of the specified limb of the 3D object. Specifying a limb number of zero provides access to the objects own root data, and should not normally be used in this way. The parameters should be specified using integer values.

SYNTAX
Return Float=LIMB DIRECTION Y(Object Number, Limb Number)

LIMB DIRECTION Z
This command will return the real number world Z angle of the specified limb of the 3D object. Specifying a limb number of zero provides access to the objects own root data, and should not normally be used in this way. The parameters should be specified using integer values.

SYNTAX
Return Float=LIMB DIRECTION Z(Object Number, Limb Number)

LIMB POSITION X
This command will return the real number world X position of the specified limb of the 3D object. Specifying a limb number of zero provides access to the objects own root data, and should not normally be used in this way. The parameters should be specified using integer values.

SYNTAX

Return Float=LIMB POSITION X(Object Number, Limb Number)

LIMB POSITION Y
This command will return the real number world Y position of the specified limb of the 3D object. Specifying a limb number of zero provides access to the objects own root data, and should not normally be used in this way. The parameters should be specified using integer values.

SYNTAX

Return Float=LIMB POSITION Y(Object Number, Limb Number)

LIMB POSITION Z
This command will return the real number world Z position of the specified limb of the 3D object. Specifying a limb number of zero provides access to the objects own root data, and should not normally be used in this way. The parameters should be specified using integer values.

SYNTAX

Return Float=LIMB POSITION Z(Object Number, Limb Number)

CHECK LIMB LINK
This command will return a value of one if a newly created limb can be added to the specified limb. You must specify the object number and limb number using integer values.

SYNTAX

Return Integer=CHECK LIMB LINK(Object Number, Limb Number)

STATIC COLLISION EXPRESSIONS

STATIC LINE OF SIGHT
This command will return a one if the specified 3D line from Sx,Sy,Sz to Dx,Dy,Dz collides with an already defined static collision box. The Width value allows you to vary the scope of detection, so that a thick line will ensure that the point at the start of the line has a very clear view of the destination point. The Accuracy value describes how accurate the returned line of sight data should be. Larger accuracy values means the line of sight calculation is quicker, but less accurate. A smaller value denotes more accuracy but the command takes longer to perform. All parameter values must be specified using real numbers.

SYNTAX

Return Integer=STATIC LINE OF SIGHT(X1, Y1, Z1, X2, Y2, Z2, Width, Accuracy)

STATIC LINE OF SIGHT X
This command will return a real value containing the X position if the command STATIC LINE OF SIGHT() returns a one. This value of one specifies that the 3D line of sight has hit a static box. You can use this value to position an object or other 3D entity at the point at which the line hit the obstruction.

SYNTAX

Return Float=STATIC LINE OF SIGHT X()

STATIC LINE OF SIGHT Y
This command will return a real value containing the Y position if the command STATIC LINE OF SIGHT() returns a one. This value of one specifies that the 3D line of sight has hit a static box. You can use this value to position an object or other 3D entity at the point at which the line hit the obstruction.

SYNTAX

Return Float=STATIC LINE OF SIGHT Y()

STATIC LINE OF SIGHT Z
This command will return a real value containing the Z position if the command STATIC LINE OF SIGHT() returns a one. This value of one specifies that the 3D line of sight has hit a static box. You can use this value to position an object or other 3D entity at the point at which the line hit the obstruction.

SYNTAX

Return Float=STATIC LINE OF SIGHT Z()

GET STATIC COLLISION HIT
You can use this command to detect whether a specified area within your 3D world is touching a static collision box. By specifying old and recent positions of the specified area, this command not only returns a one if a collision has occurred, but fills the get static collision x,y and z expressions with adjustment values. Applying these values to an object or entity will give the impression of sliding collision.

SYNTAX

Return Integer=GET STATIC COLLISION HIT(OX1, OY1, OZ1, OX2, OY2, OZ2, NX1, NY1, NZ1, NX2, NY2, NZ2)

GET STATIC COLLISION X
This command will return sliding data for X.

SYNTAX

Return Float=GET STATIC COLLISION X()

GET STATIC COLLISION Y
This command will return sliding data for Y.

SYNTAX

Return Float=GET STATIC COLLISION Y()

GET STATIC COLLISION Z
This command will return sliding data for Z.

SYNTAX

Return Float=GET STATIC COLLISION Z()

SCREEN OBJECT EXPRESSIONS

OBJECT IN SCREEN
This command will return a value of one if the specified 3D object is wholly or partly visible within the screen borders, otherwise zero is returned. Even if the object is behind the camera, its overall size may partially clip the screen area. The parameter should be specified using an integer value.

SYNTAX

Return Integer=OBJECT IN SCREEN(Object Number)

OBJECT SCREEN X
This command will return the current X screen coordinate of the specified 3D object, even if the object is not actually within the borders of the screen. You can use this for providing such things as text labels for your 3D objects or adding lens flare to a nearby street lamp. The parameter should be specified using an integer value.

SYNTAX

Return Integer=OBJECT SCREEN X(Object Number)

OBJECT SCREEN Y
This command will return the current Y screen coordinate of the specified 3D object, even if the object is not actually within the borders of the screen. You can use this for providing such things as text labels for your 3D objects or adding lens flare to a nearby street lamp. The parameter should be specified using an integer value.

SYNTAX

Return Integer=OBJECT SCREEN Y(Object Number)

PICK OBJECT
This command will return the Object Number of the closest object at the specified screen coordinates. The objects that are tested against the 2D coordinate are only those within the Object Start and Object End range in order to speed up specific tests. If no object exists at the coordinate, a value of zero is returned. If an object number is returned, additional data will be generated and stored internally. This extra data can be retrieved using the GET PICK VECTOR and GET PICK DISTANCE commands.

SYNTAX

```
Return Integer=PICK OBJECT(Screen X, Screen Y, Object Start, Object End)
```

PICK SCREEN
This command will calculate a relative 3D coordinate from a 2D screen coordinate. The additional Distance parameter indicates how far into the screen the 3D coordinate is to be calculated. The resulting 3D coordinate is not a world position, but a relative 3D vector from the current camera position. The world coordinate can be obtained by adding the camera position to the resulting vector. The resulting vector is generated internally and can be retrieved using the GET PICK VECTOR commands.

SYNTAX

```
PICK SCREEN Screen X, Screen Y, Distance
```

GET PICK DISTANCE
This command will return the Distance value from previously generated pick data. The pick data is generated when the PICK OBJECT and PICK SCREEN commands are used.

SYNTAX

```
Return Float=GET PICK DISTANCE()
```

GET PICK VECTOR X
This command will return the X offset from the vector stored in previously generated pick data. The pick data is generated when the PICK OBJECT and PICK SCREEN commands are used. The vector is relative to the world position of the camera, and so adding the camera position to this vector will result in the 3D world position.

SYNTAX

```
Return Float=GET PICK VECTOR X()
```

GET PICK VECTOR Y
This command will return the Y offset from the vector stored in previously generated pick data. The pick data is generated when the PICK OBJECT and PICK SCREEN commands are used. The vector is relative to the world position of the camera, and so adding the camera position to this vector will result in the 3D world position.

SYNTAX

```
Return Float=GET PICK VECTOR Y()
```

GET PICK VECTOR Z
This command will return the Z offset from the vector stored in previously generated pick data. The pick data is generated when the PICK OBJECT and PICK SCREEN commands are used. The vector is relative to the world position of the camera, and so adding the camera position to this vector will result in the 3D world position.

SYNTAX

```
Return Float=GET PICK VECTOR Z()
```

AVAILABILITY EXPRESSIONS

ALPHABLENDING AVAILABLE
This command will return an integer value of 1 if the current 3D card supports alphablending. Alphablending is used to create the effect of 3D semi-transparency used by the GHOST OBJECT command.

SYNTAX

```
Return Integer=ALPHABLENDING AVAILABLE()
```

ALPHA AVAILABLE
This command will return a value of one if the 3D device supports operations to the alpha channels contained within your textures and surfaces.

SYNTAX
```
Return Integer=ALPHA AVAILABLE()
```

ALPHACOMPARISON AVAILABLE
This command will return a value of one if the 3D device supports alpha comparisons, otherwise zero.

SYNTAX
```
Return Integer=ALPHACOMPARISON AVAILABLE()
```

ANISTROPICFILTERING AVAILABLE
This command will return a value of one if the 3D device supports anisotropic filtering on textures, otherwise zero.

SYNTAX
```
Return Integer=ANISTROPICFILTERING AVAILABLE()
```

ANTIALIAS AVAILABLE
This command will return a value of one if the 3D device supports anti aliasing of texture edges, otherwise zero.

SYNTAX
```
Return Integer=ANTIALIAS AVAILABLE()
```

BLITSYSTOLOCAL AVAILABLE
This command will return a value of one if the 3D device supports blitting from system to local memory, otherwise zero.

SYNTAX
```
Return Integer=BLITSYSTOLOCAL AVAILABLE()
```

CALIBRATEGAMMA AVAILABLE
This command will return a value of one if the 3D device supports modification of the display gamma, otherwise zero.

SYNTAX
```
Return Integer=CALIBRATEGAMMA AVAILABLE()
```

CLIPANDSCALEPOINTS AVAILABLE
This command will return a value of one if the 3D device supports slipping and scaling of points, otherwise zero.

SYNTAX
```
Return Integer=CLIPANDSCALEPOINTS AVAILABLE()
```

CLIPTLVERTS AVAILABLE
This command will return a value of one if the 3D device supports clipping of TL vertices otherwise zero.

SYNTAX
```
Return Integer=CLIPTLVERTS AVAILABLE()
```

COLORPERSPECTIVE AVAILABLE
This command will return a value of one if the 3D device supports color perspective correction of textures, otherwise zero.

SYNTAX
```
Return Integer=COLORPERSPECTIVE AVAILABLE()
```

COLORWRITEENABLE AVAILABLE
This command will return a value of one if the 3D device supports color write enable, otherwise zero.

SYNTAX
Return Integer=COLORWRITEENABLE AVAILABLE()

CUBEMAP AVAILABLE
This command will return a value of one if the 3D device supports cube mapping, otherwise zero.

SYNTAX
Return Integer=CUBEMAP AVAILABLE()

CULLCCW AVAILABLE
This command will return a value of one if the 3D device supports counter clockwise culling of polygons, otherwise zero.

SYNTAX
Return Integer=CULLCCW AVAILABLE()

CULLCW AVAILABLE
This command will return a value of one if the 3D device supports clockwise culling of polygons, otherwise zero.

SYNTAX
Return Integer=CULLCW AVAILABLE()

GET DEVICE TYPE
This command will return the device type of the current 3D device. This device type is returned as an integer value.

SYNTAX
Return Integer=GET DEVICE TYPE()

DITHER AVAILABLE
This command will return a value of one if the 3D device supports texture dithering, otherwise zero.

SYNTAX
Return Integer=DITHER AVAILABLE()

FILTERING AVAILABLE
This command will return an integer value of 1 if the current 3D card supports texture filtering. Texture filtering is used to smooth out your textures, creating a slight bluring effect that improves visual quality.

SYNTAX
Return Integer=FILTERING AVAILABLE()

FOGRANGE AVAILABLE
This command will return a value of one if the 3D device supports the fog range feature, otherwise zero.

SYNTAX
Return Integer=FOGRANGE AVAILABLE()

FOGTABLE AVAILABLE
This command will return a value of one if the 3D device supports the fog table method of producing fog, otherwise zero.

SYNTAX
Return Integer=FOGTABLE AVAILABLE()

FOGVERTEX AVAILABLE
This command will return a value of one if the 3D device supports the fog vertex method of producing fog, otherwise zero.

SYNTAX
Return Integer=FOGVERTEX AVAILABLE()

FOG AVAILABLE
This command will return an integer value of 1 if the current 3D card supports fogging. Fogging is used to create the effect of 3D fog used by the commands FOG ON, FOG DISTANCE and FOG COLOR.

SYNTAX
Return Integer=FOG AVAILABLE()

FULLSCREENGAMMA AVAILABLE
This command will return a value of one if the 3D device supports the fullscreen modification of the display gamma setting, otherwise zero.

SYNTAX
Return Integer=FULLSCREENGAMMA AVAILABLE()

GET MAXIMUM LIGHTS
This command will return the maximum number of lights you can have in any 3D scene. It is determined by the 3D device you are currently using.

SYNTAX
Return Integer=GET MAXIMUM LIGHTS()

GET MAXIMUM PIXEL SHADER VALUE
This command will return the maximum pixel shader value allowed by the current 3D device.

SYNTAX
Return Integer=GET MAXIMUM PIXEL SHADER VALUE()

GET MAXIMUM TEXTURE HEIGHT
This command will return the maximum texture height allowed by the current 3D device.

SYNTAX
Return Integer=GET MAXIMUM TEXTURE HEIGHT()

GET MAXIMUM TEXTURE WIDTH
This command will return the maximum texture width allowed by the current 3D device.

SYNTAX
Return Integer=GET MAXIMUM TEXTURE WIDTH()

GET MAXIMUM VERTEX SHADER CONSTANTS
This command will return the maximum number of vertex shader constants allowed by the current 3D device.

SYNTAX
Return Integer=GET MAXIMUM VERTEX SHADER CONSTANTS()

GET MAXIMUM VOLUME EXTENT
This command will return the maximum volume extent allowed by the current 3D device.

SYNTAX
Return Integer=GET MAXIMUM VOLUME EXTENT()

MIPCUBEMAP AVAILABLE
This command will return a value of one if the 3D device supports mipmapped cube mapping, otherwise zero.

SYNTAX

```
Return Integer=MIPCUBEMAP AVAILABLE()
```

MIPMAP AVAILABLE
This command will return a value of one if the 3D device supports mipmapping, otherwise zero.

SYNTAX

```
Return Integer=MIPMAP AVAILABLE()
```

MIPMAPLODBIAS AVAILABLE
This command will return a value of one if the 3D device supports a lod bias factor in the mipmapping feature, otherwise zero.

SYNTAX

```
Return Integer=MIPMAPLODBIAS AVAILABLE()
```

MIPMAPVOLUME AVAILABLE
This command will return a value of one if the 3D device supports mipmapped volumes, otherwise zero.

SYNTAX

```
Return Integer=MIPMAPVOLUME AVAILABLE()
```

NONPOWTEXTURES AVAILABLE
This command will return a value of one if the 3D device supports non-power of 2 texture sizes, otherwise zero.

SYNTAX

```
Return Integer=NONPOWTEXTURES AVAILABLE()
```

PERSPECTIVETEXTURES AVAILABLE
This command will return a value of one if the 3D device supports perspective correct texturing, otherwise zero.

SYNTAX

```
Return Integer=PERSPECTIVETEXTURES AVAILABLE()
```

GET MAXIMUM PIXEL SHADER VERSION
This command will return the maximum pixel shader version allowed by the current 3D device. The version number is returned as a real value.

SYNTAX

```
Return Float=GET MAXIMUM PIXEL SHADER VERSION()
```

PROJECTEDTEXTURES AVAILABLE
This command will return a value of one if the 3D device supports projected textures, otherwise zero.

SYNTAX

```
Return Integer=PROJECTEDTEXTURES AVAILABLE()
```

RENDERAFTERFLIP AVAILABLE
This command will return a value of one if the 3D device supports rendering to the display device directly after a flip has occurred, otherwise zero.

SYNTAX

```
Return Integer=RENDERAFTERFLIP AVAILABLE()
```

RENDERWINDOWED AVAILABLE
This command will return a value of one if the 3D device supports rendering to a window, otherwise zero.

SYNTAX
Return Integer=RENDERWINDOWED AVAILABLE()

SEPERATETEXTUREMEMORIES AVAILABLE
This command will return a value of one if the 3D device supports separate texture memories, otherwise zero.

SYNTAX
Return Integer=SEPERATETEXTUREMEMORIES AVAILABLE()

ONLYSQUARETEXTURES AVAILABLE
This command will return a value of one if the 3D device only supports square textures, otherwise zero.

SYNTAX
Return Integer=ONLYSQUARETEXTURES AVAILABLE()

TNL AVAILABLE
This command will return a value of one if the current device uses hardware TNL. TNL stands for Transformation and Lighting. Most cards these days support TNL in hardware as standard.

SYNTAX
Return Integer=TNL AVAILABLE()

TLVERTEXSYSTEMMEMORY AVAILABLE
This command will return a value of one if the 3D device supports transformation and lighting in system memory, otherwise zero.

SYNTAX
Return Integer=TLVERTEXSYSTEMMEMORY AVAILABLE()

TLVERTEXVIDEOMEMORY AVAILABLE
This command will return a value of one if the 3D device supports transformation and lighting in video memory, otherwise zero.

SYNTAX
Return Integer=TLVERTEXVIDEOMEMORY AVAILABLE()

NONLOCALVIDEOMEMORY AVAILABLE
This command will return a value of one if the 3D device supports nonlocal video memory, otherwise zero.

SYNTAX
Return Integer=NONLOCALVIDEOMEMORY AVAILABLE()

TEXTURESYSTEMMEMORY AVAILABLE
This command will return a value of one if the 3D device supports texturing from system memory, otherwise zero.

SYNTAX
Return Integer=TEXTURESYSTEMMEMORY AVAILABLE()

TEXTUREVIDEOMEMORY AVAILABLE
This command will return a value of one if the 3D device supports texturing from video memory, otherwise zero.

SYNTAX

Return Integer=TEXTUREVIDEOMEMORY AVAILABLE()

GET MAXIMUM VERTEX SHADER VERSION
This command will return the maximum vertex shader version allowed by the current 3D device. The version number is returned as a real value.

SYNTAX
Return Float=GET MAXIMUM VERTEX SHADER VERSION()

VOLUMEMAP AVAILABLE
This command will return a value of one if the 3D device supports volume mapping, otherwise zero.

SYNTAX
Return Integer=VOLUMEMAP AVAILABLE()

WBUFFER AVAILABLE
This command will return a value of one if the 3D device supports a W buffer, otherwise zero.

SYNTAX
Return Integer=WBUFFER AVAILABLE()

WFOG AVAILABLE
This command will return a value of one if the 3D device supports W Fogging, otherwise zero.

SYNTAX
Return Integer=WFOG AVAILABLE()

ZBUFFER AVAILABLE
This command will return a value of one if the 3D device supports a Z buffer, otherwise zero.

SYNTAX
Return Integer=ZBUFFER AVAILABLE()

ZFOG AVAILABLE
This command will return a value of one if the 3D device supports Z Fogging, otherwise zero.

SYNTAX
Return Integer=ZFOG AVAILABLE()

MESH EXIST
This command will return a one if the specified mesh exists, otherwise zero will be returned. The parameter should be specified using an integer value.

SYNTAX
Return Integer=MESH EXIST(Mesh Number)

EFFECT EXIST
This command will return a value of one if the specified FX effect exists. If an attempt was made to LOAD EFFECT and the load fails, no runtime error is presented. Use this command to determine if an effect was loaded successfully. Given the nature of FX files, failures can occur often when loading so use of this command is highly recommended.

SYNTAX
Return Integer=EFFECT EXIST(Effect Number)

VERTEX SHADER EXIST
This command will return a value of one if the specified vertex shader exists, otherwise zero.

SYNTAX

Return Integer=VERTEX SHADER EXIST(VertexShader Number)

PIXEL SHADER EXIST
This command will return a value of one if the specified pixel shader exists, otherwise zero.

SYNTAX

Return Integer=PIXEL SHADER EXIST(PixelShader Number)

STATISTIC
This command will return an internal statistic from the engine. Providing a value of one as the parameter will cause the command to return the current number of polygons used to render the scene.

SYNTAX

Return Integer=STATISTIC(Statistic Code)

MATRIX COMMAND SET

These commands provide functionality to control 3D world space matrix landscapes. Used in combination with the other 3D command sets, you can create, position, set and delete matrix landscapes within your application. A matrix landscape is a grid of polygons that can be terraformed into a landscape of your choice.

MAKE MATRIX
This command will create a matrix to a given width and depth segmented into grid square of a specified arrangement. A matrix can be thought of as a grid landscape with individual tiles that can be textured, raised and lowered to create a wide variety of surfaces. The matrix number and segment values should be integer values. The width and depth should be real numbers.

SYNTAX

```
MAKE MATRIX Matrix Number, Width, Height, XSegments, ZSegments
```

DELETE MATRIX
This command will delete a specified matrix previously created with make matrix. The matrix number should be an integer value.

SYNTAX

```
DELETE MATRIX Matrix Number
```

PREPARE MATRIX TEXTURE
This command will select an image of tiled textures that the matrix will eventually use. Each grid square in the matrix can have a tile texture, located within the image. Individual tile textures can be obtained from the single image by slicing it into sections both across and down. The tile textures are then assigned a number starting in the top left corner of the sectioned image and working across, then down. To section an image into 4 smaller tile textures you would specify 2 across and 2 down.

SYNTAX

```
PREPARE MATRIX TEXTURE Matrix Number, Image Number, Across, Down
```

POSITION MATRIX
This command will place the specified matrix at a position in 3D space. The matrix number should be an integer value. The coordinates should be real numbers.

SYNTAX

```
POSITION MATRIX Matrix Number, X, Y, Z
POSITION MATRIX Matrix Number, Vector
```

FILL MATRIX
This command will set each grid square in the matrix to a specified height and tile number. The matrix number should be an integer value. The height should be a real number. The tile number must be a valid texture tile allocated by the PREPARE MATRIX TEXTURE command and should be an integer value.

SYNTAX

```
FILL MATRIX Matrix Number, Height, Tile Number
```

RANDOMIZE MATRIX
This command will set each grid square in the matrix to a random height between 0 and the height value given. The matrix number should be an integer value. The height should be a real number.

SYNTAX

```
RANDOMIZE MATRIX Matrix Number, Maximum Height
```

GHOST MATRIX ON

This command will ghost a matrix. Ghosted matrices will appear transparent when rendered, creating effects such as a see-through surfaces. A ghosted matrix automatically switches to a higher level of priority when rendered in order to ensure objects above and below the matrix are visible. The Ghost Mode with a range of 0 to 5 specify the type of ghosting to perform.

SYNTAX

```
GHOST MATRIX ON Matrix Number
GHOST MATRIX ON Matrix Number, Mode
```

GHOST MATRIX OFF
This command will deactivate ghosting of a matrix. Ghosted matrices will appear transparent when rendered, creating effects such as a see-through surface.

SYNTAX

```
GHOST MATRIX OFF Matrix Number
```

SET MATRIX WIREFRAME ON
This command will set the specified matrix to display itself in wireframe form. The matrix number should be an integer value.

SYNTAX

```
SET MATRIX WIREFRAME ON Matrix Number
```

SET MATRIX WIREFRAME OFF
This command will set the specified matrix to display itself in textured form. The matrix number should be an integer value.

SYNTAX

```
SET MATRIX WIREFRAME OFF Matrix Number
```

SET VECTOR3 TO MATRIX POSITION
This command will set the vector3 data using the X, Y and Z coordinates from the specified matrix position.

SYNTAX

```
SET VECTOR3 TO MATRIX POSITION Vector, Matrix Number
```

SET MATRIX TEXTURE
This command will set different texture modes used by the specified matrix. Every texture is painted onto a matrix using an internal set of values called UV data. This data contains a range of real numbers from zero to one. Zero specifying the top/left corner of your texture and one being the bottom/right corner of your texture. When a matrix uses UV data greater and less than this range, you are permitted a number of texture wrap modes to describe what should happen to paint these areas. Setting the Texture Wrap Mode to zero will use the default wrap mode which repeats the pattern of the texture over and over, a mode of one will mirror the texture to create a seamless texture pattern and a mode of two will set clamping which retains the colour of the last pixel at the textures edge and paint with that throughout the out of range area. The Mipmap Generation Flag is used to ensure the image has a mipmap texture. A mipmap is a texture that has many levels of detail, which the matrix can select and use based on the matrix vertex distance from the camera. Use integer values to specify the parameters.

SYNTAX

```
SET MATRIX TEXTURE Matrix Number, Texture Mode, Mip Mode
```

SET MATRIX
This command will change the visual properties of the matrix. When the wireframe flag is set to 1, the matrix only shows it's wireframe form. When the transparency flag is set to 1, all parts of the matrix colored black are not drawn to the screen. When the cull flag is set to 0, the matrix will draw polygons normally hidden due to the direction the polygon faces. The Filter Value sets the texture filtering, which controls the smoothing effect of the texture as it is mapped to the object. A Filter value of zero does no mipmapping, a value of one uses no smoothing and a value of two uses Linear Filtering. The Light Flag activates and deactivates the matrices sensitivity to any lights in the scene. The Fog Flag activates and deactivates the matrices sensitivity to fog in the scene. The Ambient Flag activates and deactivates the matrices sensitivity to ambient light in the scene. The matrix number and flag values should be specified using integer values.

SYNTAX

```
SET MATRIX Matrix Number, Wire, Transparency, Cull, Filter, Light, Fog, Ambient
```

SET MATRIX TRIM
This command will set the texture tile trim value of a matrix. The trim is a float value and determines how much of the edge of the texture tile is trimmed from the final rendered output. The texture tile is the visible tile that is drawn to a grid tile of the matrix, and can sometimes suffer from the 'ugly lines' scenario when filtering causes pixels from neighboring texture tiles to be rendered. The trim value ensures these pixels can be ignored by moving the area to be used away from the neighboring texture tile.

SYNTAX
```
SET MATRIX TRIM Matrix Number, TrimX, TrimY
```

SET MATRIX PRIORITY
This command will change the priority at which the matrix is rendered to the screen. By default the matrix priority is zero and the matrix is rendered before objects, particles and other 3D elements. Changing the priority to one will cause the matrix to be drawn after all 3D elements instead, allowing the ability to render a ghosted matrix over an object that may be submerged partly within the matrix.

SYNTAX
```
SET MATRIX PRIORITY Matrix Number, Priority Flag
```

SHIFT MATRIX DOWN
This command will shift the entire contents of the matrix one grid square down. The shift ensures that the height and tile data that represent the matrix contents are wrapped around to allow continuous shifting of the landscape. The matrix number should be an integer value.

SYNTAX
```
SHIFT MATRIX DOWN Matrix Number
```

SHIFT MATRIX LEFT
This command will shift the entire contents of the matrix one grid square to the left. The shift ensures that the height and tile data that represent the matrix contents are wrapped around to allow continuous shifting of the landscape. The matrix number should be an integer value.

SYNTAX
```
SHIFT MATRIX LEFT Matrix Number
```

SHIFT MATRIX RIGHT
This command will shift the entire contents of the matrix one grid square to the right. The shift ensures that the height and tile data that represent the matrix contents are wrapped around to allow continuous shifting of the landscape. The matrix number should be an integer value.

SYNTAX
```
SHIFT MATRIX RIGHT Matrix Number
```

SHIFT MATRIX UP
This command will shift the entire contents of the matrix one grid square up. The shift ensures that the height and tile data that represent the matrix contents are wrapped around to allow continuous shifting of the landscape. The matrix number should be an integer value.

SYNTAX
```
SHIFT MATRIX UP Matrix Number
```

SET MATRIX HEIGHT
This command will set the individual height of a point within the matrix. To raise a whole grid square you would need to raise four points, one from each corner of the square. If you set points x=0 z=0, x=0 z=1, x=1 z=0 and x=1 z=1 to a value of 10, you would raise the near left square upwards by ten units. The matrix number and tile reference values should be integer values. The height value should be a real number.

SYNTAX
```
SET MATRIX HEIGHT Matrix Number, TileX, TileZ, Height
```

SET MATRIX NORMAL
This command will set the individual normal of a point within the matrix. The normal is a projected direction away from the vertex position that instructs the matrix how to take light for that point. You can use matrix normals to affect how the matrix is lit and at what strength. The matrix number and tile reference values should be integer values. The normal values should be real numbers.

SYNTAX
```
SET MATRIX NORMAL Matrix Number, TileX, TileZ, NX, NY, NZ
```

SET MATRIX TILE
This command will texture an individual grid square with an image specified by the tile number. Only if the matrix has been prepared with a texture will this command work. The tile number equates to a portion of graphic within the sectioned image you used to prepare the matrix texture. If you had prepared a matrix texture with four segmented images, you would reference these images as tile numbers from 1 to 4. The parameters should be integer values.

SYNTAX
```
SET MATRIX TILE Matrix Number, TileX, TileZ, Tile Number
```

UPDATE MATRIX
This command will update all changes you have made to an existing matrix. Any changes you have made are not visible until you complete the process by using the update matrix command. Updating the matrix is speed intensive and should be used as little as possible. The matrix number should be an integer value.

SYNTAX
```
UPDATE MATRIX Matrix Number
```

MATRIX EXIST
This command will return a one if the specified matrix exists otherwise zero will be returned. The matrix number should be an integer value.

SYNTAX
```
Return Integer=MATRIX EXIST(Matrix Number)
```

MATRIX POSITION X
This command will return the X position of the specified matrix in 3D space. The matrix number should be an integer value.

SYNTAX
```
Return Float=MATRIX POSITION X(Matrix Number)
```

MATRIX POSITION Y
This command will return the Y position of the specified matrix in 3D space. The matrix number should be an integer value.

SYNTAX
```
Return Float=MATRIX POSITION Y(Matrix Number)
```

MATRIX POSITION Z
This command will return the Z position of the specified matrix in 3D space. The matrix number should be an integer value.

SYNTAX
```
Return Float=MATRIX POSITION Z(Matrix Number)
```

MATRIX TILE COUNT
This command will return the number of available tile textures prepared for the specified matrix. The matrix number should be an integer value.

SYNTAX

Return Integer=MATRIX TILE COUNT(Matrix Number)

MATRIX TILES EXIST
This command will return a one if the matrix has been prepared with textures, otherwise zero will be returned. The matrix number should be an integer value.

SYNTAX

Return Integer=MATRIX TILES EXIST(Matrix Number)

MATRIX WIREFRAME STATE
This command will return a one if the specified matrix is displayed as a wireframe otherwise zero will be returned. The matrix number should be an integer value.

SYNTAX

Return Integer=MATRIX WIREFRAME STATE(Matrix Number)

GET GROUND HEIGHT
This command will calculate and return the Y coordinate within the matrix given the X and Z coordinates. This command can be used to allows 3D objects to traverse the contours of any matrix landscape with ease. The matrix number should be an integer value. The coordinates should be real numbers.

SYNTAX

Return Float=GET GROUND HEIGHT(Matrix Number, X, Z)

GET MATRIX HEIGHT
This command will return the matrix height at the specified grid point coordinate. Do not confuse the grid reference X and Z values with coordinate values. The grid reference values are measured per tile, not per 3D space unit. The matrix number and grid reference values should be integer values.

SYNTAX

Return Float=GET MATRIX HEIGHT(Matrix Number, TileX, TileZ)

WORLD COMMAND SET

These commands provide functionality to control 3D BSP and Terrain worlds. Used in combination with the other 3D command sets, you can create, position, set and delete worlds within your application. A BSP world is a predefined description of a world containing 3D geometry, textures and collision information. A Terrain world is an advanced form of matrix lanscape that can be created from a heightmap stored in a simple picture file.

LOAD BSP
This command will load a BSP world into the 3D scene. The BSP world automatically draws to the current camera and cannot be rotated or repositioned. Culling, texturing and collision are all handled automatically and through the related world commands. The first parameter allows you to specify a packed file containing the BSP world files, normally a PAK or PK3 file. If the BSP world is not being loaded from a packed file, this parameter should be an empty string. The second parameter is the BSP filename of the world you wish to load. This file can either be contained within the specified packed file, or exist as a standalone BSP file. If loading the BSP file as a standalone file, ensure the current directory contains any files used by the BSP world.

SYNTAX
```
LOAD BSP PK3 Filename, BSP Filename
```

DELETE BSP
This command will delete a BSP world from the 3D scene. All data loaded with the LOAD BSP command will be removed from the system allowing you to load a new BSP world.

SYNTAX
```
DELETE BSP
```

SET BSP CAMERA
This command will set the main camera for the BSP system. The BSP camera is used to calculate how to crop the BSP polygons to keep the performance level high. Binary Space Partitioning works by drawing only the polygons currently visible from the perspective of the specified camera.

SYNTAX
```
SET BSP CAMERA Camera Number
```

SET BSP CAMERA COLLISION
This command will set the collision sphere of the camera for the BSP world. This will automatically adjust the position of the camera when it attempts to enter solid BSP geometry. The camera will be repositioned to produce the smoothest collision response. If the Response value is set to one, the new position backtracks to the last collision free position when a hit occurs. It is worth noting that even commands that reposition the camera are subject to the BSP collision once active. To avoid problems repositioning the camera, simply deactivate and reactivate the collision during positioning.

SYNTAX
```
SET BSP CAMERA COLLISION Collision Index, Camera Number, Radius, Response
```

SET BSP OBJECT COLLISION
This command will set the collision sphere of the specified object for the BSP world. This will automatically adjust the position of the object when it attempts to enter solid BSP geometry. The object will be repositioned to produce the smoothest collision response. If the Response value is set to one, the new position backtracks to the last collision free position when a hit occurs. It is worth noting that even commands that reposition the object are subject to the BSP collision once active. To avoid problems repositioning your objects, simply deactivate and reactivate the collision during positioning.

SYNTAX
```
SET BSP OBJECT COLLISION Collision Index, Object Number, Radius, Response
```

SET BSP COLLISION THRESHHOLD
This command will set the sensitivity of the auto-adjustment effect of the automated collision activated by the SET BSP OBJECT COLLISION and SET BSP CAMERA COLLISION commands. The default is zero, which causes the collision to

slowly affect the entity from even the smallest pull of negative Y movement. The effect is a slow slide down uneven surfaces. Increasing the sensitivity value will provide a range under which this sliding effect does not occur.

SYNTAX

SET BSP COLLISION THRESHHOLD Collision Index, Sensitivity

SET BSP COLLISION OFF
This command will remove the collision system assigned to the BSP collision index previously assigned a collision type using the command SET BSP OBJECT COLLISION or SET BSP CAMERA COLLISION.

SYNTAX

SET BSP COLLISION OFF Collision Index

SET BSP CAMERA COLLISION RADIUS
This command will set the BSP camera collision radius used during collision detection. You can use this command in combination with the SET BSP COLLISION HEIGHT ADJUSTMENT to fine tune your collision entity within the BSP universe.

SYNTAX

SET BSP CAMERA COLLISION RADIUS Collision Index, Camera Number, X, Y, Z

SET BSP OBJECT COLLISION RADIUS
This command will set the BSP object collision radius used during collision detection. You can use this command in combination with the SET BSP COLLISION HEIGHT ADJUSTMENT to fine tune your collision entity within the BSP universe.

SYNTAX

SET BSP OBJECT COLLISION RADIUS Collision Index, Object Number, X, Y, Z

SET BSP COLLISION HEIGHT ADJUSTMENT
This command will set an offset used when positioning the collision sphere used during BSP collision. This can be useful for when the mass of your collision area of the object is off center from the mathematical center of your entity, or when you would like to move your camera off the floor of a BSP world to give your first person player some height.

SYNTAX

SET BSP COLLISION HEIGHT ADJUSTMENT Collision Index, Height

SET BSP MULTITEXTURING ON
This command will switch on multitexturing within a BSP rendered world. This is the default behaviour.

SYNTAX

SET BSP MULTITEXTURING ON

SET BSP MULTITEXTURING OFF
This command will switch off multitexturing within a BSP rendered world.

SYNTAX

SET BSP MULTITEXTURING OFF

PROCESS BSP COLLISION
This command will calculate the effect the BSP collision has on the respective camera or object. Normally the BSP collision effects are calculated and applied during the refresh, however with this command you can process the effect immediately. This can be useful when you wish to obtain the new position of a collided object or camera position for additional calculations such as associating a carried item to the location of a player or enemy character.

SYNTAX

PROCESS BSP COLLISION Collision Index

MAKE TERRAIN
This command will create a new terrain object using the specified Heightmap file. A heightmap file is a simple image file that contains a greyscale representation of the landscape heights. The image file must contain an image with identical width and height, and ideally be a resolution equal or greater than 256 colours. Additional parameters control the size and detail of the terrain.

SYNTAX

```
MAKE TERRAIN Terrain Number, Heightmap Filename
```

DELETE TERRAIN
This command will delete a terrain object previously created with the MAKE TERRAIN command.

SYNTAX

```
DELETE TERRAIN Terrain Number
```

POSITION TERRAIN
This command will position the specified terrain object in 3D world space. Repositioning the terrain will not affect the relative coordinates used by the GET TERRAIN HEIGHT command, which is always based on the current position of the terrain.

SYNTAX

```
POSITION TERRAIN Terrain Number, X, Y, Z
```

TEXTURE TERRAIN
This command will texture the specified terrain object using a preloaded image. The image is used to texture the terrain, and is tiled repeatedly over the terrain to create a seamless layer of detail to the terrain floor.

SYNTAX

```
TEXTURE TERRAIN Terrain Number, Image Number
```

BSP COLLISION HIT
This command will return a value of one if the specified collision index hit solid BSP geometry during the last rendering cycle. Use this command in combination with the BSP COLLISION X, Y and Z to return sliding collision data from the collision event.

SYNTAX

```
Return Integer=BSP COLLISION HIT(Collision Index)
```

BSP COLLISION X
This command will return an X coordinate modifier value when the specified collision index is involved in a hit against solid BSP geometry during the rendering cycle. Use this command in combination with the BSP COLLISION HIT command to determine when the hit occurs.

SYNTAX

```
Return Float=BSP COLLISION X(Collision Index)
```

BSP COLLISION Y
This command will return an Y coordinate modifier value when the specified collision index is involved in a hit against solid BSP geometry during the rendering cycle. Use this command in combination with the BSP COLLISION HIT command to determine when the hit occurs.

SYNTAX

```
Return Float=BSP COLLISION Y(Collision Index)
```

BSP COLLISION Z
This command will return an Z coordinate modifier value when the specified collision index is involved in a hit against solid BSP geometry during the rendering cycle. Use this command in combination with the BSP COLLISION HIT command to determine when the hit occurs.

SYNTAX

Return Float=BSP COLLISION Z(Collision Index)

TERRAIN EXIST
This command will return a value of one if the terrain object exists, otherwise zero.

SYNTAX
Return Integer=TERRAIN EXIST(Terrain Number)

TERRAIN POSITION X
This command will return the X coordinate of the specified terrain object.

SYNTAX
Return Float=TERRAIN POSITION X(Terrain Number)

TERRAIN POSITION Y
This command will return the Y coordinate of the specified terrain object.

SYNTAX
Return Float=TERRAIN POSITION Y(Terrain Number)

TERRAIN POSITION Z
This command will return the Z coordinate of the specified terrain object.

SYNTAX
Return Float=TERRAIN POSITION Z(Terrain Number)

GET TERRAIN HEIGHT
This command will return the height of the terrain at the specified coordinate of the terrain. The coordinates are specified using real numbers and are based relative to the position of the terrain. The position of the terrain does not affect the coordinates specified here.

SYNTAX
Return Float=GET TERRAIN HEIGHT(Terrain Number, X, Z)

GET TOTAL TERRAIN HEIGHT
This command will return the overall maximum height of the terrain landscape. A terrain feature that was produced from a white pixel of the heightmap is the highest point on the terrain, and will match exactly the value returned here.

SYNTAX
Return Float=GET TOTAL TERRAIN HEIGHT(Terrain Number)

PARTICLES COMMAND SET

These commands provide functionality to control 3D world space particle objects. Used in combination with the other 3D command sets, you can create, position, point, set and delete particle objects within your application. A particle object emits particles under a user defined behaviour and can be used for fire, snow, explosions, smoke and any effect that requires the use of many small particle entities.

MAKE PARTICLES
This command will create a particles object using a specified image and radius. A particles object will emit single particles given a default set of rules than can be changed using the particle commands.

SYNTAX

MAKE PARTICLES Particle Number, Image Number, Frequency, Radius

MAKE SNOW PARTICLES
This command will create a particles object and set it to act like a snow drift. The area specified by the position and size is identical to that of a box created with the same dimensions.

SYNTAX

MAKE SNOW PARTICLES Particle Number, Image Number, Frequency, X, Y, Z, Width, Height, Depth

MAKE FIRE PARTICLES
This command will create a particles object and set it to act like fire particles. The area specified by the position and size is identical to that of a box created with the same dimensions.

SYNTAX

MAKE FIRE PARTICLES Particle Number, Image Number, Frequency, X, Y, Z, Width, Height, Depth

DELETE PARTICLES
This command will delete a particles object from memory. If you simply wish to hide an object, it is suggested you reposition or hide the particles object as recreating particle objects from scratch has a performance hit.

SYNTAX

DELETE PARTICLES Particle Number

SHOW PARTICLES
This command will show a particles object, and will instantly show every particle which is deemed part of the particles object previously hidden with the HIDE PARTICLES command.

SYNTAX

SHOW PARTICLES Integer Value

HIDE PARTICLES
This command will hide a particles object, and will instantly hide every particle which is deemed part of the particles object. To emit no new particles and leave the rest to descend, either reduce the number of emissions or reposition the particles object offscreen.

SYNTAX

HIDE PARTICLES Particle Number

POSITION PARTICLES
This command will position a particles object in 3D world space. By moving a particles object you will be moving at the same time every particle that belongs to the particles object. To create the effect of trails, see POSITION PARTICLE EMISSIONS command.

SYNTAX

```
POSITION PARTICLES Particle Number, X, Y, Z
POSITION PARTICLES Particle Number, Vector
```

POSITION PARTICLE EMISSIONS
This command will adjust the emissions position of a particles object. By moving the emissions position of the particles object, you can create very effective visual trails.

SYNTAX

```
POSITION PARTICLE EMISSIONS Particle Number, X, Y, Z
```

ROTATE PARTICLES
This command will rotate a particles object. Rotating the particles object will rotate every particle currently existing as part of the particles object. To correct the behaviour of the individual particles themselves, refer to the other particles object commands.

SYNTAX

```
ROTATE PARTICLES Particle Number, X, Y, Z
ROTATE PARTICLES Particle Number, Vector
```

SET VECTOR3 TO PARTICLES POSITION
This command will set the vector3 data using the X, Y and Z coordinates from the specified particles position.

SYNTAX

```
SET VECTOR3 TO PARTICLES POSITION Vector, Particle Number
```

SET VECTOR3 TO PARTICLES ROTATION
This command will set the vector3 data using the X, Y and Z angles from the specified particles rotation.

SYNTAX

```
SET VECTOR3 TO PARTICLES ROTATION Vector, Particle Number
```

COLOR PARTICLES
This command will set the color of new particles emitted from a particles object. By using a generally white image, the color you specify will show up the best from the particles. An image that is not white will produce a blending of colour between the image and the specified color.

SYNTAX

```
COLOR PARTICLES Particle Number, Red, Green, Blue
```

SET PARTICLE EMISSIONS
This command will set the number of particles emitted from the particles object each cycle. A value of zero will stop the particles object emitting new particles.

SYNTAX

```
SET PARTICLE EMISSIONS Particle Number, Frequency
```

SET PARTICLE VELOCITY
This command will set the velocity of particles within the particles object. This controls the velocity at which the particles will move once emitted, and reflect the amount of energy expelled from the particles object.

SYNTAX

```
SET PARTICLE VELOCITY Particle Number, Velocity Value
```

SET PARTICLE GRAVITY
This command will set the gravity of particles within the particles object. This controls the actual course of the particle. A negative gravity value will actually pull the particle upwards at a rotation of 0,0,0. A high positive gravity value will make the particle very heavy. The default is a gravity value of 5.

SYNTAX

SET PARTICLE GRAVITY Particle Number, Gravity Value

SET PARTICLE SPEED
This command will set the number of seconds that are to pass each frame within the particles object. This controls whether the particles move very quickly, or very slowly. The default seconds per frame value is 0.005.

SYNTAX

SET PARTICLE SPEED Particle Number, Seconds Per Frame

SET PARTICLE CHAOS
This command will set the chaos behaviour of particles within the particles object. This controls the stability of the particles position in 3D world space. The default is zero. A value greater or less than zero will vibrate the particle by the magnitude of the chaos value.

SYNTAX

SET PARTICLE CHAOS Particle Number, Chaos Value

SET PARTICLE FLOOR
This command will set whether the particle within the particles object will hit a predetermined floor at zero Y in 3D world space. The default is a flag value of one. A value of zero will remove the invisible floor allowing particles to continue to exist until they eventually fade.

SYNTAX

SET PARTICLE FLOOR Particle Number, Floor Flag

SET PARTICLE LIFE
This command will set the life duration of particles within the particles object. This controls how long the particle will stay alive. The default value of 100 keeps the particle alive long enough to arc, hit the floor and spark. A value smaller than this will stop short this existence.

SYNTAX

SET PARTICLE LIFE Particle Number, Life Percentage

GHOST PARTICLES ON
This command will ghost a particles object. Ghosted particles will appear transparent when rendered, creating effects such as see-through smoke and fire. The Ghost Mode with a range of 0 to 5 specify the type of ghosting to perform.

SYNTAX

GHOST PARTICLES ON Particle Number, Ghost Mode

GHOST PARTICLES OFF
This command will deactivate ghosting of a particles object. Ghosted particles will appear transparent when rendered, creating effects such as see-through smoke and fire.

SYNTAX

GHOST PARTICLES OFF Particle Number

PARTICLES EXIST
This command will return a value of one if the particles object exists, otherwise zero.

SYNTAX

Return Integer=PARTICLES EXIST(Particle Number)

PARTICLES POSITION X
This command will return the X position of the particles object in 3D world space.

SYNTAX

```
Return Float=PARTICLES POSITION X(Particle Number)
```

PARTICLES POSITION Y
This command will return the Y position of the particles object in 3D world space.

SYNTAX

```
Return Float=PARTICLES POSITION Y(Particle Number)
```

PARTICLES POSITION Z
This command will return the Z position of the particles object in 3D world space.

SYNTAX

```
Return Float=PARTICLES POSITION Z(Particle Number)
```

3DMATHS COMMAND SET

These commands provide functionality for advanced 3D arithmatic with the use of four additional datatypes. Vector2, Vector3, Vector4 and Matrix4 are four datatypes that are specifically used for advanced 3D maths.

ADD MATRIX4
This command will add two matrices together.

SYNTAX
```
ADD MATRIX4 Matrix4Result, Matrix4A, Matrix4B
```

ADD VECTOR2
This command will add two vectors together. This vector is defined as a two float vector.

SYNTAX
```
ADD VECTOR2 VectorResult, VectorA, VectorB
```

ADD VECTOR3
This command will add two vectors together. This vector is defined as a three float vector.

SYNTAX
```
ADD VECTOR3 VectorResult, VectorA, VectorB
```

ADD VECTOR4
This command will add two vectors together. This vector is defined as a four float vector.

SYNTAX
```
ADD VECTOR4 VectorResult, VectorA, VectorB
```

BUILD LOOKAT LHMATRIX4
This command builds a LOOKAT left handed Matrix.

SYNTAX
```
BUILD LOOKAT LHMATRIX4 Matrix4Result, Vector3Eye, Vector3At, Vector3Up
```

BUILD LOOKAT RHMATRIX4
This command builds a LOOKAT right handed Matrix.

SYNTAX
```
BUILD LOOKAT RHMATRIX4 Matrix4Result, Vector3Eye, Vector3At, Vector3Up
```

BUILD ORTHO LHMATRIX4
This command builds a Orthogonal left handed Matrix.

SYNTAX
```
BUILD ORTHO LHMATRIX4 Matrix4Result, Width, Height, Near, Far
```

BUILD ORTHO RHMATRIX4
This command builds a Orthogonal right handed Matrix.

SYNTAX
```
BUILD ORTHO RHMATRIX4 Matrix4Result, Width, Height, Near, Far
```

BUILD FOV LHMATRIX4
This command builds a Field-Of-View left handed Matrix.

SYNTAX
```
BUILD FOV LHMATRIX4 Matrix4Result, FOV, Aspect, Near, Far
```

BUILD FOV RHMATRIX4
This command builds a Field-Of-View right handed Matrix.

SYNTAX
```
BUILD FOV RHMATRIX4 Matrix4Result, FOV, Aspect, Near, Far
```

BUILD PERSPECTIVE LHMATRIX4
This command builds a Perspective left handed Matrix.

SYNTAX
```
BUILD PERSPECTIVE LHMATRIX4 Matrix4Result, Width, Height, Near, Far
```

BUILD PERSPECTIVE RHMATRIX4
This command builds a Perspective right handed Matrix.

SYNTAX
```
BUILD PERSPECTIVE RHMATRIX4 Matrix4Result, Width, Height, Near, Far
```

BUILD REFLECTION MATRIX4
This command builds a Reflection Matrix.

SYNTAX
```
BUILD REFLECTION MATRIX4 Matrix4Result, PlaneA, PlaneB, PlaneC, PlaneD
```

BUILD ROTATION AXIS MATRIX4
This command builds a Rotation Axis Matrix.

SYNTAX
```
BUILD ROTATION AXIS MATRIX4 Matrix4Result, Vector3Axis, Angle
```

CATMULLROM VECTOR2
This command performs a catmull rom interpolation on the specified vector. This vector is defined as a two float vector.

SYNTAX
```
CATMULLROM VECTOR2 VectorResult, VectorA, VectorB, VectorC, VectorD, Value
```

CATMULLROM VECTOR3
This command performs a catmull rom interpolation on the specified vector. This vector is defined as a three float vector.

SYNTAX
```
CATMULLROM VECTOR3 VectorResult, VectorA, VectorB, VectorC, VectorD, Value
```

CATMULLROM VECTOR4
This command performs a catmull rom interpolation on the specified vector. This vector is defined as a four float vector.

SYNTAX
```
CATMULLROM VECTOR4 VectorResult, VectorA, VectorB, VectorC, VectorD, Value
```

COPY MATRIX4
This command will copy a matrix.

SYNTAX
```
COPY MATRIX4 Matrix4Result, Matrix4Source
```

COPY VECTOR2
This command copy a vector. This vector is defined as a two float vector.

SYNTAX
```
COPY VECTOR2 VectorResult, VectorSource
```

COPY VECTOR3
This command copy a vector. This vector is defined as a three float vector.

SYNTAX
```
COPY VECTOR3 VectorResult, VectorSource
```

COPY VECTOR4
This command copy a vector. This vector is defined as a four float vector.

SYNTAX
```
COPY VECTOR4 VectorResult, VectorSource
```

CROSS PRODUCT VECTOR3
This command will calculate a Cross Product Vector.

SYNTAX
```
CROSS PRODUCT VECTOR3 VectorResult, VectorA, VectorB
```

DELETE MATRIX4
This command will delete a matrix.

SYNTAX
```
Return Integer=DELETE MATRIX4(Matrix4Result)
```

DELETE VECTOR2
This command will delete a vector. This is defined as a two float vector.

SYNTAX
```
Return Integer=DELETE VECTOR2(Vector)
```

DELETE VECTOR3
This command will delete a vector. This is defined as a three float vector.

SYNTAX
```
Return Integer=DELETE VECTOR3(Vector)
```

DELETE VECTOR4
This command will delete a vector. This is defined as a four float vector.

SYNTAX
```
Return Integer=DELETE VECTOR4(Vector)
```

DIVIDE MATRIX4
This command will divide a matrix.

SYNTAX

```
DIVIDE MATRIX4 Matrix4Result, Value
```

DIVIDE VECTOR2
This command will divide a vector. This is defined as a two float vector.

SYNTAX
```
DIVIDE VECTOR2 VectorResult, Value
```

DIVIDE VECTOR3
This command will divide a vector. This is defined as a three float vector.

SYNTAX
```
DIVIDE VECTOR3 VectorResult, Value
```

DIVIDE VECTOR4
This command will divide a vector. This is defined as a four float vector.

SYNTAX
```
DIVIDE VECTOR4 VectorResult, Value
```

DOT PRODUCT VECTOR2
This command returns the dot product of two vectors. This is defined as a two float vector.

SYNTAX
```
Return Float=DOT PRODUCT VECTOR2(VectorA, VectorB)
```

DOT PRODUCT VECTOR3
This command returns the dot product of two vectors. This is defined as a three float vector.

SYNTAX
```
Return Float=DOT PRODUCT VECTOR3(VectorA, VectorB)
```

BCC VECTOR2
This command produces the BaryCentricCoordinates vector. This vector is defined as a two float vector.

SYNTAX
```
BCC VECTOR2 VectorResult, VectorA, VectorB, VectorC, FValue, GBValue
```

BCC VECTOR3
This command produces the BaryCentricCoordinates vector. This vector is defined as a three float vector.

SYNTAX
```
BCC VECTOR3 VectorResult, VectorA, VectorB, VectorC, FValue, GBValue
```

BCC VECTOR4
This command produces the BaryCentricCoordinates vector. This vector is defined as a four float vector.

SYNTAX
```
BCC VECTOR4 VectorResult, VectorA, VectorB, VectorC, FValue, GBValue
```

CCW VECTOR2
This command returns the z component by taking the cross product of both vectors. This vector is defined as a two float vector.

SYNTAX
```
Return Float=CCW VECTOR2(VectorA, VectorB)
```

SQUARED LENGTH VECTOR2
This command will return the squared length of a vector. This defines as a two float vector.

SYNTAX
```
Return Float=SQUARED LENGTH VECTOR2(Vector)
```

SQUARED LENGTH VECTOR3
This command will return the squared length of a vector. This defines as a three float vector.

SYNTAX
```
Return Float=SQUARED LENGTH VECTOR3(Vector)
```

SQUARED LENGTH VECTOR4
This command will return the squared length of a vector. This defines as a four float vector.

SYNTAX
```
Return Float=SQUARED LENGTH VECTOR4(Vector)
```

LENGTH VECTOR2
This command will return the length of a vector. This is defined as a two float vector.

SYNTAX
```
Return Float=LENGTH VECTOR2(Vector)
```

LENGTH VECTOR3
This command will return the length of a vector. This is defined as a three float vector.

SYNTAX
```
Return Float=LENGTH VECTOR3(Vector)
```

LENGTH VECTOR4
This command will return the length of a vector. This is defined as a four float vector.

SYNTAX
```
Return Float=LENGTH VECTOR4(Vector)
```

PROJECTION MATRIX4
This command will copy the projection matrix into the specified matrix.

SYNTAX
```
PROJECTION MATRIX4 Matrix4Result
```

VIEW MATRIX4
This command will copy the view matrix into the specified matrix.

SYNTAX
```
VIEW MATRIX4 Matrix4Result
```

W VECTOR4
This command returns the W value from the vector.

SYNTAX
```
Return Float=W VECTOR4(Vector)
```

WORLD MATRIX4
This command will copy the world matrix into the specified matrix.

SYNTAX
WORLD MATRIX4 Matrix4Result

X VECTOR2
This command returns the X coordinate from the vector. This defines at a two float vector.

SYNTAX
Return Float=X VECTOR2(Vector)

X VECTOR3
This command returns the X coordinate from the vector. This defines at a three float vector.

SYNTAX
Return Float=X VECTOR3(Vector)

X VECTOR4
This command returns the X coordinate from the vector. This defines at a four float vector.

SYNTAX
Return Float=X VECTOR4(Vector)

Y VECTOR2
This command returns the Y coordinate from the vector. This defines at a two float vector.

SYNTAX
Return Float=Y VECTOR2(Vector)

Y VECTOR3
This command returns the Y coordinate from the vector. This defines at a three float vector.

SYNTAX
Return Float=Y VECTOR3(Vector)

Y VECTOR4
This command returns the Y coordinate from the vector. This defines at a four float vector.

SYNTAX
Return Float=Y VECTOR4(Vector)

Z VECTOR3
This command returns the Z coordinate from the vector. This defines at a three float vector.

SYNTAX
Return Float=Z VECTOR3(Vector)

Z VECTOR4
This command returns the Z coordinate from the vector. This defines at a four float vector.

SYNTAX
Return Float=Z VECTOR4(Vector)

HERMITE VECTOR2
This command performs a hermite spline interpolation on a vector. This is defined as a two float vector.

SYNTAX

HERMITE VECTOR2 VectorResult, VectorA, VectorB, VectorC, VectorD, SValue

HERMITE VECTOR3

This command performs a hermite spline interpolation on a vector. This is defined as a three float vector.

SYNTAX

HERMITE VECTOR3 VectorResult, VectorA, VectorB, VectorC, VectorD, SValue

HERMITE VECTOR4

This command performs a hermite spline interpolation on a vector. This is defined as a four float vector.

SYNTAX

HERMITE VECTOR4 VectorResult, VectorA, VectorB, VectorC, VectorD, SValue

INVERSE MATRIX4

This command will invert a matrix.

SYNTAX

Return Float=INVERSE MATRIX4(Matrix4Result, Matrix4Source)

IS EQUAL MATRIX4

This command returns a one if the matrices specified are identical, otherwise zero.

SYNTAX

Return Integer=IS EQUAL MATRIX4(Matrix4A, Matrix4B)

IS EQUAL VECTOR2

This command returns a one if the vectors specified are identical, otherwise zero. This defines as a two float vector.

SYNTAX

Return Integer=IS EQUAL VECTOR2(VectorA, VectorB)

IS EQUAL VECTOR3

This command returns a one if the vectors specified are identical, otherwise zero. This defines as a three float vector.

SYNTAX

Return Integer=IS EQUAL VECTOR3(VectorA, VectorB)

IS EQUAL VECTOR4

This command returns a one if the vectors specified are identical, otherwise zero. This defines as a four float vector.

SYNTAX

Return Integer=IS EQUAL VECTOR4(VectorA, VectorB)

IS IDENTITY MATRIX4

This command returns a one if the matrix is an identity matrix.

SYNTAX

Return Integer=IS IDENTITY MATRIX4(Matrix4Result)

LINEAR INTERPOLATE VECTOR2

This command perform a linear interpolation between 2 vectors. This is defined as a two float vector.

SYNTAX

```
LINEAR INTERPOLATE VECTOR2 VectorResult, VectorA, VectorB, SValue
```

LINEAR INTERPOLATE VECTOR3
This command perform a linear interpolation between 2 vectors. This is defined as a three float vector.

SYNTAX
```
LINEAR INTERPOLATE VECTOR3 VectorResult, VectorA, VectorB, SValue
```

LINEAR INTERPOLATE VECTOR4
This command perform a linear interpolation between 2 vectors. This is defined as a four float vector.

SYNTAX
```
LINEAR INTERPOLATE VECTOR4 VectorResult, VectorA, VectorB, SValue
```

MAKE MATRIX4
This command will create a MATRIX4 data item. Unlike the datatypes you are used to, these datatypes are stored internally and referenced by an index value. The MATRIX4 datatype contains sixteen float values representing a 4x4 matrix of data. This is a typical 3D matrix used in many advanced 3D calculations. You can imagine the grid headed with columns X,Y,Z,W by rows X,Y,Z,W.

SYNTAX
```
Return Integer=MAKE MATRIX4(Matrix4)
```

MAKE VECTOR2
This command will create a VECTOR2 data item. Unlike the datatypes you are used to, these datatypes are stored internally and referenced by an index value. The VECTOR2 datatype contains two float values representing X and Y. This datatype is typically used to store screen based coordinates.

SYNTAX
```
Return Integer=MAKE VECTOR2(Vector)
```

MAKE VECTOR3
This command will create a VECTOR3 data item. Unlike the datatypes you are used to, these datatypes are stored internally and referenced by an index value. The VECTOR3 datatype contains three float values representing X, Y and Z. This datatype is typically used to store 3D world space coordinates.

SYNTAX
```
Return Integer=MAKE VECTOR3(Vector)
```

MAKE VECTOR4
This command will create a VECTOR4 data item. Unlike the datatypes you are used to, these datatypes are stored internally and referenced by an index value. The VECTOR4 datatype contains four float values representing X, Y, Z and W. This datatype is typically used to store transformed and untransformed 3D world space coordinates.

SYNTAX
```
Return Integer=MAKE VECTOR4(Vector)
```

MAXIMIZE VECTOR2
This command will maximize a vector. This defines as a two float vector.

SYNTAX
```
MAXIMIZE VECTOR2 VectorResult, VectorA, VectorB
```

MAXIMIZE VECTOR3
This command will maximize a vector. This defines as a three float vector.

SYNTAX

```
MAXIMIZE VECTOR3 VectorResult, VectorA, VectorB
```

MAXIMIZE VECTOR4
This command will maximize a vector. This defines as a four float vector.

SYNTAX
```
MAXIMIZE VECTOR4 VectorResult, VectorA, VectorB
```

MINIMIZE VECTOR2
This command will minimize a vector. This defines as a two float vector.

SYNTAX
```
MINIMIZE VECTOR2 VectorResult, VectorA, VectorB
```

MINIMIZE VECTOR3
This command will minimize a vector. This defines as a three float vector.

SYNTAX
```
MINIMIZE VECTOR3 VectorResult, VectorA, VectorB
```

MINIMIZE VECTOR4
This command will minimize a vector. This defines as a four float vector.

SYNTAX
```
MINIMIZE VECTOR4 VectorResult, VectorA, VectorB
```

MULTIPLY MATRIX4
This command will multiply two matrices together.

SYNTAX
```
MULTIPLY MATRIX4 Matrix4Result, Matrix4A, Matrix4B
MULTIPLY MATRIX4 Matrix4Result, Value
```

MULTIPLY VECTOR2
This command will multiply two vectors together. This defines as a two float vector.

SYNTAX
```
MULTIPLY VECTOR2 VectorResult, Value
```

MULTIPLY VECTOR3
This command will multiply two vectors together. This defines as a three float vector.

SYNTAX
```
MULTIPLY VECTOR3 VectorResult, Value
```

MULTIPLY VECTOR4
This command will multiply two vectors together. This defines as a four float vector.

SYNTAX
```
MULTIPLY VECTOR4 VectorResult, Value
```

NORMALIZE VECTOR2
This command will normalize a vector. This defines as a two float vector.

SYNTAX

```
NORMALIZE VECTOR2 VectorResult, VectorSource
```

NORMALIZE VECTOR3
This command will normalize a vector. This defines as a three float vector.

SYNTAX
```
NORMALIZE VECTOR3 VectorResult, VectorSource
```

NORMALIZE VECTOR4
This command will normalize a vector. This defines as a four float vector.

SYNTAX
```
NORMALIZE VECTOR4 VectorResult, VectorSource
```

PROJECT VECTOR3
This command projects a given vector from object space into screen space. This defines as a three float vector.

SYNTAX
```
PROJECT VECTOR3 VectorResult, VectorSource, Matrix4Projection, Matrix4View, Matrix4World
```

ROTATE X MATRIX4
This command will rotate the matrix on it's x axis.

SYNTAX
```
ROTATE X MATRIX4 Matrix4Result, Angle
```

ROTATE Y MATRIX4
This command will rotate the matrix on it's y axis.

SYNTAX
```
ROTATE Y MATRIX4 Matrix4Result, Angle
```

ROTATE YPR MATRIX4
This command builds a matrix from Pitch Yaw Roll values.

SYNTAX
```
ROTATE YPR MATRIX4 Matrix4Result, Yaw, Pitch, Roll
```

ROTATE Z MATRIX4
This command will rotate the matrix on it's z axis.

SYNTAX
```
ROTATE Z MATRIX4 Matrix4Result, Angle
```

SCALE MATRIX4
This command will scale a matrix.

SYNTAX
```
SCALE MATRIX4 Matrix4Result, X, Y, Z
```

SCALE VECTOR2
This command will scale a vector. This defines as a two float vector.

SYNTAX
```
SCALE VECTOR2 VectorResult, VectorSource, Value
```

SCALE VECTOR3
This command will scale a vector. This defines as a three float vector.

SYNTAX
```
SCALE VECTOR3 VectorResult, VectorSource, Value
```

SCALE VECTOR4
This command will scale a vector. This defines as a four float vector.

SYNTAX
```
SCALE VECTOR4 VectorResult, VectorSource, Value
```

SET IDENTITY MATRIX4
This command will set the specified matrix to a standard identity matrix. Transforming a vector or matrix using an identity matrix will not produce a change in the resulting data.

SYNTAX
```
SET IDENTITY MATRIX4 Matrix4Result
```

SET VECTOR2
This command will set a vector. This defines as a two float vector.

SYNTAX
```
SET VECTOR2 VectorResult, X, Y
```

SET VECTOR3
This command will set a vector. This defines as a three float vector.

SYNTAX
```
SET VECTOR3 VectorResult, X, Y, Z
```

SET VECTOR4
This command will set a vector. This defines as a four float vector.

SYNTAX
```
SET VECTOR4 VectorResult, X, Y, Z, W
```

SUBTRACT MATRIX4
This command will subtract one matrix from another.

SYNTAX
```
SUBTRACT MATRIX4 Matrix4Result, Matrix4A, Matrix4B
```

SUBTRACT VECTOR2
This command will subtract one vector from another. This vector is defined as a two float vector.

SYNTAX
```
SUBTRACT VECTOR2 VectorResult, VectorA, VectorB
```

SUBTRACT VECTOR3
This command will subtract one vector from another. This vector is defined as a three float vector.

SYNTAX
```
SUBTRACT VECTOR3 VectorResult, VectorA, VectorB
```

SUBTRACT VECTOR4
This command will subtract one vector from another. This vector is defined as a four float vector.

SYNTAX
SUBTRACT VECTOR4 VectorResult, VectorA, VectorB

TRANSFORM VECTOR4
This command will transform a vector.

SYNTAX
TRANSFORM VECTOR4 VectorResult, VectorSource, Matrix4Source

TRANSFORM COORDS VECTOR2
This command will transform a vector. This vector is defined as a two float vector.

SYNTAX
TRANSFORM COORDS VECTOR2 VectorResult, VectorSource, Matrix4Source

TRANSFORM COORDS VECTOR3
This command will transform vector. This vector is defined as a three float vector.

SYNTAX
TRANSFORM COORDS VECTOR3 VectorResult, VectorSource, Matrix4Source

TRANSFORM NORMALS VECTOR3
This command will transform a normals vector. This vector is defined as a three float vector.

SYNTAX
TRANSFORM NORMALS VECTOR3 VectorResult, VectorSource, Matrix4Source

TRANSLATE MATRIX4
This command will produce a translation matrix.

SYNTAX
TRANSLATE MATRIX4 Matrix4Result, X, Y, Z

TRANSPOSE MATRIX4
This command will transpose a matrix.

SYNTAX
TRANSPOSE MATRIX4 Matrix4Result, Matrix4Source

FTP COMMAND SET

These commands provide functionality for accessing FTP servers. You can use these commands to upload and download files from a server on the internet you have permissions to access. Ideal for sharing data with a wider community, or something as simple as a universal hiscore table.

FTP CONNECT
This command will allow you to connect to FTP sites.

SYNTAX
```
FTP CONNECT Url String, User String, Password String
```

FTP DISCONNECT
This command will allow you to disconnect from an ftp site previously connected to using the FTP CONNECT command. You can optionally specify an integer parameter to disconnect the dial-up connection if dial-up access was used.

SYNTAX
```
FTP DISCONNECT
FTP DISCONNECT Dial-Up-Disconnect Flag
```

FTP TERMINATE
This command will allow you to terminate a current download started by FTP GET FILE.

SYNTAX
```
FTP TERMINATE
```

FTP SET DIR
This command will allow you to go to an FTP directory.

SYNTAX
```
FTP SET DIR Directory String
```

FTP GET FILE
This command will allow you to use FTP PROCEED to grab. The GrabInBits Flag allows you to specify the amount in bytes to be grabbed each time FTP PROCEED is called, thus controlling the rate and responsiveness of your application during download.

SYNTAX
```
FTP GET FILE Ftp Filename String
FTP GET FILE Ftp Filename String, Local Filename String, GrabInBits Flag
```

FTP PUT FILE
This command will allow you to copy a local file into the current ftp directory.

SYNTAX
```
FTP PUT FILE Local Filename String
```

FTP DELETE FILE
This command will allow you to delete an ftp file from the current ftp directory.

SYNTAX
```
FTP DELETE FILE Ftp Filename String
```

FTP PROCEED
This command will allow you to grab another chunk of the downloading file started by FTP GET FILE.

SYNTAX

FTP PROCEED

FTP FIND FIRST
This command will allow you to find the first ftp file in the current ftp directory.

SYNTAX

FTP FIND FIRST

FTP FIND NEXT
This command will allow you to find the next ftp file in the current ftp directory.

SYNTAX

FTP FIND NEXT

GET FTP STATUS
This command will return zero if the ftp connection has failed.

SYNTAX

Return Integer=GET FTP STATUS()

GET FTP DIR$
This command will return the current ftp directory.

SYNTAX

Return String=GET FTP DIR$()

GET FTP PROGRESS
This command will return a percentage value of the amount of the file being downloaded. When the download is complete, this command will return a value of minus one.

SYNTAX

Return Integer=GET FTP PROGRESS()

GET FTP FAILURE
This command will return a one if the ftp connection has failed to execute an ftp command. If a one was returned, you can use the GET FTP ERROR$() command to determine the actual reason for the failure.

SYNTAX

Return Integer=GET FTP FAILURE()

GET FTP ERROR$
This command will return a string containing the description of a failure from a previously called ftp command. You can determine whether an ftp command has failed by calling the GET FTP FAILURE() command.

SYNTAX

Return String=GET FTP ERROR$()

GET FTP FILE TYPE
This command will return the ftp filetype pointed to by the commands FIND FIRST/NEXT.

SYNTAX

Return Integer=GET FTP FILE TYPE()

GET FTP FILE NAME$

This command will return the ftp filename pointed to by the commands FIND FIRST/NEXT.

SYNTAX

Return String=GET FTP FILE NAME$()

GET FTP FILE SIZE

This command will return the size of the current ftp file being pointed at.

SYNTAX

Return Integer=GET FTP FILE SIZE()

MEMBLOCKS COMMAND SET

These commands provide functionality to create, modify and delete memblocks. Memblocks are blocks of memory allocated by the user to store any type of data. They are also used as a medium to store and recreate common media resources with the language including bitmaps, images, sounds and 3D meshes.

MAKE MEMBLOCK
This command will make a memblock of the given size. If the memblock already exists this command will fail. The parameters must be specified using integer values.

SYNTAX
```
MAKE MEMBLOCK Memblock Number, Size in Bytes
```

MAKE MEMBLOCK FROM BITMAP
This command will make a memblock from a bitmap. The specified values must be integer values and the source resource must exist or the command will fail. The memblock stores the data as Width(DWORD), Height(DWORD), Depth(DWORD), Data(BYTES).

SYNTAX
```
MAKE MEMBLOCK FROM BITMAP Memblock Number, Bitmap Number
```

MAKE MEMBLOCK FROM IMAGE
This command will make a memblock from an image. The specified values must be integer values and the source resource must exist or the command will fail. The memblock stores the data as Width(DWORD), Height(DWORD), Depth(DWORD), Data(BYTES).

SYNTAX
```
MAKE MEMBLOCK FROM IMAGE Memblock Number,Image Number
```

MAKE MEMBLOCK FROM SOUND
This command will make a memblock from a sound. The specified values must be integer values and the source resource must exist or the command will fail. The memblock stores the data as BitsPerSecond(DWORD), Frequency(DWORD), Duration(DWORD), Sound Data(BYTES).

SYNTAX
```
MAKE MEMBLOCK FROM SOUND Memblock Number, Sound Number
```

MAKE MEMBLOCK FROM MESH
This command will make a memblock from a mesh. The specified values must be integer values and the source resource must exist or the command will fail. The mesh memblock is layed out in the following format. The first DWORD is the FVF Format, which controls which components each vertex of your mesh will contain. The default FVF Format is 338. The second DWORD is the FVF Size, which is the size in bytes of a single vertex element. This size is respective of the FVF Format you specified, which has a default of 36. The third DWORD is the number of Vertices in your mesh. The remainder of the memblock contains mesh data. The mesh data is a sequential list of vertices, containing the component data arranged as specified by the FVF Format. The default FVF Format would specify the following arrangement of data within the vertex element, which is duplicated for every vertex specified in the memblock. Each grouping of three vertices makes a polygon. Given the default FVF Format of 338, the first three FLOAT values (12 bytes) of the vertex element would be the XYZ coordinates in model space. The second three FLOAT values (12 bytes) of the vertex element would be the normals coordinates in model space. The next DWORD is a diffuse colour component that specifies the colour of the vertex. The last two FLOATS are UV texture coordinates for the vertex. This adds up to 36 bytes which is the size of a single vertex. Multiply 36 by the number of vertices in the mesh and you get the overall size of the mesh data.

SYNTAX
```
MAKE MEMBLOCK FROM MESH Memblock Number,Mesh Number
```

DELETE MEMBLOCK
This command will delete a memblock. If the memblock does not exist this command will fail. The parameter must be specified using an integer value.

SYNTAX

```
DELETE MEMBLOCK Memblock Number
```

COPY MEMBLOCK
This command will copy one section of a memblock to another section of another memblock. The From and To parameters must be existing memblocks. The PosFrom and PosTo parameters must be byte locations within the respective memblocks. The Bytes parameter is the number of bytes you wish to copy from one memblock to the other.

SYNTAX

```
COPY MEMBLOCK From,To,PosFrom,PosTo,Bytes
```

MAKE BITMAP FROM MEMBLOCK
This command will make a bitmap from a memblock. The specified values must be integer values and the source resource must exist or the command will fail. The memblock stores the data as described in the respective memblock construction command.

SYNTAX

```
MAKE BITMAP FROM MEMBLOCK Bitmap Number,Memblock Number
```

MAKE IMAGE FROM MEMBLOCK
This command will make an image from a memblock. The specified values must be integer values and the source resource must exist or the command will fail. The memblock stores the data as described in the respective memblock construction command.

SYNTAX

```
MAKE IMAGE FROM MEMBLOCK Image Number, Memblock Number
```

MAKE SOUND FROM MEMBLOCK
This command will make a sound from a memblock. The specified values must be integer values and the source resource must exist or the command will fail. The memblock stores the data as described in the respective memblock construction command.

SYNTAX

```
MAKE SOUND FROM MEMBLOCK Sound Number, Memblock Number
```

MAKE MESH FROM MEMBLOCK
This command will make a mesh from a memblock. The specified values must be integer values and the source resource must exist or the command will fail. The mesh memblock is layed out in the following format. The first DWORD is the FVF Format, which controls which components each vertex of your mesh will contain. The default FVF Format is 338. The second DWORD is the FVF Size, which is the size in bytes of a single vertex element. This size is respective of the FVF Format you specified, which has a default of 36. The third DWORD is the number of Vertices in your mesh. The remainder of the memblock contains mesh data. The mesh data is a sequential list of vertices, containing the component data arranged as specified by the FVF Format. The default FVF Format would specify the following arrangement of data within the vertex element, which is duplicated for every vertex specified in the memblock. Each grouping of three vertices makes a polygon. Given the default FVF Format of 338, the first three FLOAT values (12 bytes) of the vertex element would be the XYZ coordinates in model space. The second three FLOAT values (12 bytes) of the vertex element would be the normals coordinates in model space. The next DWORD is a diffuse colour component that specifies the colour of the vertex. The last two FLOATS are UV texture coordinates for the vertex. This adds up to 36 bytes which is the size of a single vertex. Multiply 36 by the number of vertices in the mesh and you get the overall size of the mesh data.

SYNTAX

```
MAKE MESH FROM MEMBLOCK Mesh Number, Memblock Number
```

CHANGE MESH FROM MEMBLOCK
This command will change a mesh from a memblock. This command is similar to MAKE MESH FROM MEMBLOCK, and skips the mesh creation step to make this command faster. You must ensure the mesh uses the same data structure as the memblock or this command will produce undesirable results. The specified values must be integer values and the source resource must exist or the command will fail. The memblock stores the data as described in the respective memblock construction command.

SYNTAX

```
CHANGE MESH FROM MEMBLOCK Mesh Number,Memblock Number
```

WRITE MEMBLOCK BYTE
This command will write a byte into the specified location of the memblock. The memblock must exist or the command will fail. The Position value is specified in bytes. The Byte value must be a value between 0 and 255. The parameters must be specified using integer values.

SYNTAX

```
WRITE MEMBLOCK BYTE Memblock Number, Position, Byte
```

WRITE MEMBLOCK WORD
This command will write a word into the specified location of the memblock. A Word is the term for a datatype consisting of two bytes. The memblock must exist or the command will fail. The Position value is specified in bytes. The Word must be a value between 0 and 65535. The parameters must be specified using integer values.

SYNTAX

```
WRITE MEMBLOCK WORD Memblock Number, Position, Word
```

WRITE MEMBLOCK DWORD
This command will write a dword into the specified location of the memblock. A DWord is the term for a datatype consisting of four bytes. The memblock must exist or the command will fail. The Position value is specified in bytes. The DWord value must be a value between 0 and 2147483648. The parameters must be specified using integer values.

SYNTAX

```
WRITE MEMBLOCK DWORD Memblock Number, Position, DWord
```

WRITE MEMBLOCK FLOAT
This command will write a float into the specified location of the memblock. A Float is the term for a datatype consisting of four bytes, also referred to as a real number. The memblock must exist or the command will fail. The Position value is specified in bytes. The Float value must be a value between -2147483648 to 2147483648. The parameters must be specified using integer values.

SYNTAX

```
WRITE MEMBLOCK FLOAT Memblock Number, Position, Float
```

MEMBLOCK EXIST
This command will return a value of one if the specified memblock exists, otherwise zero is returned. The parameter must be an integer value.

SYNTAX

```
Return Integer=MEMBLOCK EXIST(Memblock Number)
```

GET MEMBLOCK PTR
This command will return the actual pointer of the specified memblock. You can use this pointer to pass into a DLL in order to access the memory directly. The parameter must be an integer value.

SYNTAX

```
Return DWORD=GET MEMBLOCK PTR(Memblock Number)
```

GET MEMBLOCK SIZE
This command will return the size of the specified memblock. The size will be returned in bytes. The parameter must be an integer value.

SYNTAX

```
Return Integer=GET MEMBLOCK SIZE(Memblock Number)
```

MEMBLOCK BYTE

This command will read a byte from the specified location of the memblock. The memblock must exist or the command will fail. The Position value is specified in bytes. The Byte returned will be a value between 0 and 255. The parameters must be specified using integer values.

SYNTAX

Return Integer=MEMBLOCK BYTE(Memblock Number, Position)

MEMBLOCK WORD
This command will read a word from the specified location of the memblock. The memblock must exist or the command will fail. The Position value is specified in bytes. The Word returned will be a value between 0 and 65535. The parameters must be specified using integer values.

SYNTAX

Return Integer=MEMBLOCK WORD(Memblock Number, Position)

MEMBLOCK DWORD
This command will read a dword from the specified location of the memblock. The memblock must exist or the command will fail. The Position value is specified in bytes. The DWord returned will be a value between 0 and 2147483648. The parameters must be specified using integer values.

SYNTAX

Return DWORD=MEMBLOCK DWORD(Memblock Number, Position)

MEMBLOCK FLOAT
This command will read a float from the specified location of the memblock. The memblock must exist or the command will fail. The Position value is specified in bytes. The Float returned will be a value between -2147483648 to 2147483648. The parameters must be specified using integer values.

SYNTAX

Return Float=MEMBLOCK FLOAT(Memblock Number, Position)

MULTIPLAYER COMMAND SET

These commands provide functionality to create a multiplayer game. The commands have been designed to make the creation, joining, handling and freeing of net games simple. You can exchange all forms of data, including memblocks and media resources such as images, sounds and 3D meshes.

DEFAULT NET GAME

This command will automatically set up a net game in one command. It will create a game if non exists on the most suitable network connection, and join a game that already exists. It will select a LAN connection first, then a TCP/IP connection and then Serial or Modem connection. You can specify both a Game name and a Player name, as well as the number of players allowed in the game session. You can also specify a flag that determines the kind of game that will be created. Specify a game type of 1 to make the game pass host control to another player if the host leaves play. Specify a game type value of 2 if you want to create a client server game where the host player channels all communication to the other players. The command also returns the Player Number that you can use to distinguish your player from other network players.

SYNTAX

```
Return Integer=DEFAULT NET GAME(Gamename, Playername, Number Of Players, Game Type)
```

CREATE NET GAME

This command will create a multiplayer net game. The Gamename describes the name of the game and will be the name of the session when the game begins. The Playername is the name you wish to give the initial host of the game. The number of players can be between 2 and 256, and sets the maximum number of players that can join the net game. You can optionally specify a flag which controls the type of net game created. A value of 1 is the default and specifies a Pier to Pier game, where a value of 2 specifies a Client/Server game. A Pier to Pier game has every computer communicate with each other. A Client/Server game has all player traffic routed through the host computer and then broadcasted to the rest of the players. You will need to set a connection before you can create a net game, however this command will automatically select the first connection it finds if you have selected one.

SYNTAX

```
CREATE NET GAME Gamename, Playername, Number Of Players
CREATE NET GAME Gamename, Playername, Number Of Players, Flag
```

JOIN NET GAME

This command will join a currently running net game session. To join a net game, you must first find one using PERFORM CHECKLIST FOR NET SESSIONS. From this you can find the Session Number required to join the net game. The Playername is the name the player uses when entering the net game session. If the specified session does not exist, or there are too many players in the session, this command will fail.

SYNTAX

```
JOIN NET GAME Session Number, Playername
```

FREE NET GAME

This command will terminate a net game currently in session. You can only have one net game running at any one time per application, so in order to create a new game you must first free up any existing net games currently running. If you joined an existing net game, this command will remove you from the game and the game will continue running without you.

SYNTAX

```
FREE NET GAME
```

CREATE NET PLAYER

This command will create another player within the net game. This player will be an additional player to the default player created when you created or joined the game. You can use this command if you wished to populate your net game with allies or enemies to be treated like regular players. The Playername is the given name of the player for the net game. You can optionally return the Player Number at the moment of creation.

SYNTAX

```
CREATE NET PLAYER Playername
Return Integer=CREATE NET PLAYER(Playername)
```

FREE NET PLAYER
This command will remove a player from the current net game. The Player Number refers to the Unique ID that was given to the player when it was created. You can obtain this value from the checklist value a result produced when you use the PERFORM CHECKLIST FOR NET PLAYERS command.

SYNTAX

```
FREE NET PLAYER Player Number
```

SET NET CONNECTION
This command will set the machine to a specific connection. The Connection Number can be obtained by using the index of the checklist produced by the PERFORM CHECKLIST FOR NET CONNECTIONS command. You can optionally specify Address Data depending on the connection type. If you connect by TCP/IP the Address Data should be an IP and Url Address. If the connection type is MODEM, you should specify a phone number if you are dialling or leave blank if you are answering. If the connection type is SERIAL, it is recommended you leave the Address Data blank to obtain the Windows Serial Configuration Dialogue Box. IPX connections require no additional Address Data.

SYNTAX

```
SET NET CONNECTION Connection Number
SET NET CONNECTION Connection Number,Address Data
```

PERFORM CHECKLIST FOR NET CONNECTIONS
This command will fill the checklist with the names of all the currently available connections on the machine. There are usually four types of connections available to you including TCP/IP, IPX, Modem and Serial. TCP/IP is used to connect via an IP Address and is used for Internet games. IPX is used for playing net games over a Local Area Network (LAN). A Modem connection is a two player net game with a Dial player and a Receive player, played over a phone line. A Serial connection is a direct cable connecting two machines, and acts much like a Modem connection. The index of the checklist is also the connection number associated with the connection description obtained from this command. Use the CHECKLIST commands in the SYSTEM command set to read the checklist.

SYNTAX

```
PERFORM CHECKLIST FOR NET CONNECTIONS
```

PERFORM CHECKLIST FOR NET SESSIONS
This command will fill the checklist with the names of all the currently available sessions on the previously specified connection. The session names represent a currently running net game. These are net games you are able to join. The index of the checklist is also the session number associated with the session description obtained from this command. Use the CHECKLIST commands in the SYSTEM command set to read the checklist.

SYNTAX

```
PERFORM CHECKLIST FOR NET SESSIONS
```

PERFORM CHECKLIST FOR NET PLAYERS
This command will fill the checklist with all the players currently seen by the currently active net game. The checklist contains five pieces of data for each player listed. The checklist string contains the given name of the player. The checklist value A contains a Unique ID provided for the player when the player appeared in the net game. This ID is only unique to the application and will remain with the player as long as it resides in the net game. You can use this ID to reference an array containing the players game data. The checklist value B contains a special universal ID for the player, and this value does not change from machine to machine. You can use it to isolate a player on any machine. Checklist Value C will be set to one if the listed player is the current local player. Checklist Value D will be set to one if the listed player is the host player of the net game. Use the CHECKLIST commands in the SYSTEM command set to read the checklist.

SYNTAX

```
PERFORM CHECKLIST FOR NET PLAYERS
```

SEND NET MESSAGE INTEGER
This command will send a message containing an integer value to the specified player. The Player Number must be an integer value and an existing player in the net game. A Player Number of zero will send the message to all players except you.

SYNTAX

```
SEND NET MESSAGE INTEGER Player Number, Integer Value
```

SEND NET MESSAGE FLOAT
This command will send a message containing a float value to the specified player. The Player Number must be an integer value and an existing player in the net game. A Player Number of zero will send the message to all players except you.

SYNTAX
```
SEND NET MESSAGE FLOAT Player Number, Float Value
```

SEND NET MESSAGE STRING
This command will send a message containing a string to the specified player. The Player Number must be an integer value and an existing player in the net game. A Player Number of zero will send the message to all players except you.

SYNTAX
```
SEND NET MESSAGE STRING Player Number, String
```

SEND NET MESSAGE MEMBLOCK
This command will send a message containing a memblock to the specified player. The Player Number must be an integer value and an existing player in the net game. The memblock must exist or the command will fail. A Player Number of zero will send the message to all players except you. The Guarantee Packet Flag, if set to one, will ensure the message is received and will not be dropped due to slow system performance.

SYNTAX
```
SEND NET MESSAGE MEMBLOCK Player Number, Memblock Number
SEND NET MESSAGE MEMBLOCK Player Number, Memblock Number, Guarentee Packet
```

SEND NET MESSAGE BITMAP
This command will send a message containing a bitmap to the specified player. The Player Number must be an integer value and an existing player in the net game. A Player Number of zero will send the message to all players except you. The Guarantee Packet Flag, if set to one, will ensure the message is received and will not be dropped due to slow system performance.

SYNTAX
```
SEND NET MESSAGE BITMAP Player Number, Bitmap Number, Guarentee Packet
```

SEND NET MESSAGE IMAGE
This command will send a message containing an image to the specified player. The Player Number must be an integer value and an existing player in the net game. A Player Number of zero will send the message to all players except you. The Guarantee Packet Flag, if set to one, will ensure the message is received and will not be dropped due to slow system performance.

SYNTAX
```
SEND NET MESSAGE IMAGE Player Number, Image Number, Guarentee Packet
```

SEND NET MESSAGE SOUND
This command will send a message containing a sound to the specified player. The Player Number must be an integer value and an existing player in the net game. A Player Number of zero will send the message to all players except you. The Guarantee Packet Flag, if set to one, will ensure the message is received and will not be dropped due to slow system performance.

SYNTAX
```
SEND NET MESSAGE SOUND Player Number, Sound Number, Guarentee Packet
```

SEND NET MESSAGE MESH
This command will send a message containing a mesh to the specified player. The Player Number must be an integer value and an existing player in the net game. A Player Number of zero will send the message to all players except you. The Guarantee Packet Flag, if set to one, will ensure the message is received and will not be dropped due to slow system performance.

SYNTAX

SEND NET MESSAGE MESH Player Number, Mesh Number, Guarentee Packet

GET NET MESSAGE
This command gets the oldest message from the incoming message queue and makes it the current message. Any messages that are sent to this application are stored on a queue and you are able to take each message and process it. You can use the NET MESSAGE EXISTS() command to determine when there are no more messages in the queue.

SYNTAX

GET NET MESSAGE

NET MESSAGE INTEGER
This command will return an integer value from the current net message. The net message must be of the integer type or the command will fail. You can determine the type using the NET MESSAGE TYPE command.

SYNTAX

Return Integer=NET MESSAGE INTEGER()

NET MESSAGE FLOAT
This command will return a float value from the current net message. The net message must be of the float type or the command will fail. You can determine the type using the NET MESSAGE TYPE command.

SYNTAX

Return Float=NET MESSAGE FLOAT()

NET MESSAGE STRING$
This command will return a string from the current net message. The net message must be of the string type or the command will fail. You can determine the type using the NET MESSAGE TYPE command.

SYNTAX

Return String=NET MESSAGE STRING$()

NET MESSAGE MEMBLOCK
This command will return a memblock from the current net message. The net message must be of the memblock type or the command will fail. You can determine the type using the NET MESSAGE TYPE command. If a memblock already exists under this number, that memblock is deleted and the new memblock is created in its place.

SYNTAX

NET MESSAGE MEMBLOCK Memblock Number

NET MESSAGE IMAGE
This command will return an image from the current net message. The net message must be of the image type or the command will fail. You can determine the type using the NET MESSAGE TYPE command.

SYNTAX

NET MESSAGE IMAGE Image Number

NET MESSAGE BITMAP
This command will return a bitmap from the current net message. The net message must be of the bitmap type or the command will fail. You can determine the type using the NET MESSAGE TYPE command.

SYNTAX

NET MESSAGE BITMAP Bitmap Number

NET MESSAGE SOUND
This command will return a sound from the current net message. The net message must be of the sound type or the command will fail. You can determine the type using the NET MESSAGE TYPE command.

SYNTAX

NET MESSAGE SOUND Sound Number

NET MESSAGE MESH
This command will return a mesh from the current net message. The net message must be of the mesh type or the command will fail. You can determine the type using the NET MESSAGE TYPE command.

SYNTAX

NET MESSAGE MESH Mesh Number

NET GAME EXISTS
This command will return a value of one if the net game exists, otherwise zero is returned.

SYNTAX

Return Integer=NET GAME EXISTS()

NET MESSAGE EXISTS
This command will return a value of one if the message queue contains one or more messages for the current application. A value zero means the queue is empty.

SYNTAX

Return Integer=NET MESSAGE EXISTS()

NET MESSAGE PLAYER FROM
This command returns the Player Number that the current message was sent from. You can use this to determine who sent the message.

SYNTAX

Return Integer=NET MESSAGE PLAYER FROM()

NET MESSAGE PLAYER TO
This command returns the Player Number that the current message is being sent to. You can use this to determine who the message is for, and whether it is a message for you.

SYNTAX

Return Integer=NET MESSAGE PLAYER TO()

NET MESSAGE TYPE
This command returns the type of the current message in the queue. The type can be one of four values. If the type is 1, the message is an integer value. If the type is 2, the message is a float value. If the type is 3, the message is a string. If the type is a 4, then the message is a memblock. If the type is a 5, the message is an image. If the type is a 6, the message is a bitmap. If the type is a 7, the message is a sound. If the type is an 8, the message is a mesh. You must check the type of a message before you can read it correctly using one of the appropriate NET MESSAGE commands.

SYNTAX

Return Integer=NET MESSAGE TYPE()

NET GAME LOST
This command will return a value of one if the net game has been lost, otherwise zero is returned.

SYNTAX

Return Integer=NET GAME LOST()

NET GAME NOW HOSTING
This command will return a value of one if the application has just been made the host of the net game, otherwise zero is returned. During a Pier to Pier net game, if the host leaves the net game then host status is migrated to another player in the net game.

SYNTAX

Return Integer=NET GAME NOW HOSTING()

NET BUFFER SIZE
This command will return the number of items queuing in the net buffer to be processed. This buffer can fill up if there is too much data coming into the net game. You can use this value to regulate the amount of traffic your program generates thus reducing this value. An ideal count is zero.

SYNTAX

Return Integer=NET BUFFER SIZE()

NET PLAYER DESTROYED
This command will return the Player Number of a newly removed player from the net game. The Player Number is the Unique ID you have been using to reference this player. You do not need to free this player when you receive this signal as it has already been done for you.

SYNTAX

Return Integer=NET PLAYER DESTROYED()

NET PLAYER CREATED
This command will return the Player Number of a newly created player to the net game. The Player Number is the Unique ID you can use to initialise a new player in your game data. You do not need to create a new player when you receive this signal as it has already been done for you.

SYNTAX

Return Integer=NET PLAYER CREATED()

SYSTEM COMMAND SET

These commands provide functionality to control and monitor the system properties of your application. In addition, you are able to access checklist commands which are filled with information when a suitable command is called. Checklists are powerful general systems to collect information on all aspects of your system. You are also able to load, call and delete DLLs which provides instant expandability to the language.

LOAD DLL
This command will load a DLL into memory under the specified DLL Number. The DLL file must exist either in the current working directory or the Windows system folder. If the file is not found this command will fail. The DLL Number must be an integer value between 1 and 256.

SYNTAX

```
LOAD DLL DLLName, DLL Number
```

DELETE DLL
This command will delete a previously loaded DLL. If the DLL does not exist this command will fail. The DLL Number must be an integer value between 1 and 256.

SYNTAX

```
DELETE DLL DLL Number
```

CALL DLL
This command will call a function of a loaded DLL. The DLL Number must be an integer value between 1 and 256. The DLL Number points to the DLL previously loaded. The Function String is the name of the function described in the export table of the DLL. You can optionally have up to 9 parameters of integer, real or string type providing the function you are calling matches the parameters exactly. You can optionally return a value of integer, real or string type providing the function exports the same type.

SYNTAX

```
CALL DLL DLL Number, Function Name, Parameters List
```

DLL EXIST
This command determines whether a DLL has been loaded successfully. The DLL Number must be an integer value between 1 and 256. If the DLL exists an integer value of one is returned, otherwise zero.

SYNTAX

```
Return Integer=DLL EXIST(DLL Number)
```

DLL CALL EXIST
This command determines whether a function call exists within a loaded DLL. The DLL Number must be an integer value between 1 and 256. The DLL Number specifies a previously loaded DLL and the Function String describes the function name within the DLL. If the function exists an integer value of one is returned, otherwise zero.

SYNTAX

```
Return Integer=DLL CALL EXIST(DLL Number, Function Name)
```

EMPTY CHECKLIST
This command will clear the general purpose checklist facility.

SYNTAX

```
EMPTY CHECKLIST
```

CHECKLIST QUANTITY
This command will return the total number of items in the checklist after a PERFORM CHECKLIST command has been performed. The checklist has a maximum storage capacity of 255 items.

SYNTAX

```
Return Integer=CHECKLIST QUANTITY()
```

CHECKLIST STRING$
This command will return the string from the specified item number in the checklist after a PERFORM CHECKLIST command has been performed. The item number should be an integer value.

SYNTAX

```
Return String=CHECKLIST STRING$(Integer Value)
```

CHECKLIST VALUE A
This command will return value A from the specified item number in the checklist after a PERFORM CHECKLIST command has been performed. The item number should be an integer value.

SYNTAX

```
Return Integer=CHECKLIST VALUE A(Integer Value)
```

CHECKLIST VALUE B
This command will return value B from the specified item number in the checklist after a PERFORM CHECKLIST command has been performed. The item number should be an integer value.

SYNTAX

```
Return Integer=CHECKLIST VALUE B(Integer Value)
```

CHECKLIST VALUE C
This command will return value C from the specified item number in the checklist after a PERFORM CHECKLIST command has been performed. The item number should be an integer value.

SYNTAX

```
Return Integer=CHECKLIST VALUE C(Integer Value)
```

CHECKLIST VALUE D
This command will return value D from the specified item number in the checklist after a PERFORM CHECKLIST command has been performed. The item number should be an integer value.

SYNTAX

```
Return Integer=CHECKLIST VALUE D(Integer Value)
```

DISABLE ESCAPEKEY
This command will deactivate the escape key. Be aware that this command will prevent breaking back into your program without a suitable exit condition.

SYNTAX

```
DISABLE ESCAPEKEY
```

ENABLE ESCAPEKEY
This command will reactivate the escape key. The escape key is deactivated using the DISABLE ESCAPEKEY.

SYNTAX

```
ENABLE ESCAPEKEY
```

DISABLE SYSTEMKEYS
This command will disable the use of such system keys as ALT+TAB, ALT+F4. This command will fail if the operating system does not support the screen saver based method of deactivating the system keys.

SYNTAX

```
DISABLE SYSTEMKEYS
```

ENABLE SYSTEMKEYS
This command will enable all system keys disabled with DISABLE SYSTEMKEYS.

SYNTAX
```
ENABLE SYSTEMKEYS
```

EXIT PROMPT
This command will create a message box when the application exits. You can specify the text for the title and the text area of the message box. You can use this command to prompt that your application is a demo version or use it to report debug information in your standalone executables.

SYNTAX
```
EXIT PROMPT Message String, Caption String
```

SYSTEM TMEM AVAILABLE
This command will return the total memory available on your system.

SYNTAX
```
Return Integer=SYSTEM TMEM AVAILABLE()
```

SYSTEM DMEM AVAILABLE
This command will return the total video memory available on the system.

SYNTAX
```
Return Integer=SYSTEM DMEM AVAILABLE()
```

SYSTEM SMEM AVAILABLE
This command will return the total system memory available on the system.

SYNTAX
```
Return Integer=SYSTEM SMEM AVAILABLE()
```

INDEX